Segurança na arquitetura TCP/IP:
de firewalls a canais seguros

Keesje Duarte Pouw

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Segurança na arquitetura TCP/IP: de firewalls a canais seguros

Keesje Duarte Pouw[1]

Janeiro de 1999

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (Orientador)
- Profa. Dra. Liane Margarida Rockenbach Tarouco
  DEE, Faculdade de Educação, UFRGS
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, UNICAMP
- Prof. Dr. Célio Cardoso Guimarães (Suplente)
  Instituto de Computação, UNICAMP

1. financiado por CNPq e FAPESP

Ficha Catalográfica elaborada pela Biblioteca Central da UNICAMP

Pouw, Keesje Duarte
P868s Segurança na arquitetura TCP/IP: de firewalls a canais seguros / Keesje Duarte Pouw. – Campinas, SP: [s.n.], 1999.

Orientador: Paulo Lício de Geus.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Internet (Redes de computação) - Medidas de segurança. 2. TCP/IP (Protocolos de redes de computação) - Medidas de segurança. 3. Redes de computação - Protocolos - Medidas de segurança. 4. Criptografia. 5. UNIX (Sistema operacional de computador) - Medidas de segurança. 6. Computadores - Medidas de segurança. 7. World Wide Web (Sistema de recuperação de informação) - Medidas de segurança. I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título

Segurança na arquitetura TCP/IP: de firewalls a canais seguros

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Keesje Duarte Pouw e aprovada pela Banca Examinadora.

Campinas, janeiro de 1999

Paulo Lício de Geus (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Dissertação defendida e aprovada em 08 de fevereiro de 1999, pela Banca Examinadora composta pelos Professores Doutores:

Porfa. Dra. Liane Margarida Rockenbach Tarouco
UFRGS

Prof. Dr. Ricardo Dahab
IC - UNICAMP

Prof. Dr. Paulo Lício de Geus
IC - UNICAMP

Resumo

O espectro de protocolos e aplicações TCP/IP é provavelmente tão amplo quanto é premente a necessidade de mecanismos de segurança nesta arquitetura. O alcance deste aspecto de segurança vai desde a segurança intrínseca de sistemas operacionais, passando por *firewalls*, criptografia e canais seguros, até aplicações específicas. De maneira geral, poucas referências tratam este problema de maneira completa, especialmente no que se refere à criptografia e sua relação com os protocolos TCP/IP seguros.

Desta forma, o foco principal deste trabalho é enumerar os problemas inerentes à arquitetura TCP/IP, tratando as questões de segurança com especial ênfase na tecnologia de *firewalls* e nos algoritmos de criptografia, assim como na sua relação na construção de canais seguros de comunicação. Como aplicação deste estudo, obteve-se diretrizes para o desenvolvimento de aplicações críticas do ponto de vista de segurança em ambiente HTML/HTTPS, as quais serviram de base para a implementação do produto de Internet banking do BankBoston no Brasil.

Abstract

The range of TCP/IP protocols and applications is probably as wide as is strong the need for security mechanisms in this architecture. The scope of the security aspects goes from the inherent security of operating systems, through *firewalls*, cryptography and secure channels, up to specific applications. As a general rule, little literature deals thoroughly with this issue, especially with regards to cryptography and its relation to secure TCP/IP protocols.

As such, this work focuses on enumerating the intrinsic problems of the TCP/IP architecture, dealing with the security issues with special emphasis in *firewall* technology and in cryptographic algorithms, as well as in their relation with regards to building secure communication channels. As an application for this study, directions for the development of critical security applications in an HTML/HTTPS environment were obtained, which were the basis for the implementation of the Internet banking system for BankBoston in Brazil.

Dedico aos meus pais, meus grandes amigos, companheiros e incentivadores, sem os quais este trabalho não teria sido realizado.

Agradecimentos

Ao professor Paulo Lício de Geus, por toda orientação, apoio e atenção fornecidas durante o desenvolvimento deste trabalho.

Ao professor Ricardo Dahab pela dedicação e apoio no desenvolvimento deste trabalho.

Aos colegas do Instituto de Computação, em especial a José Roberto Menezes Monteiro, pela amizade, aprendizado e incentivo mútuo no desenvolvimento de cada pesquisa.

Ao CNPq e FAPESP pela bolsas de estudo concedidas.

À Luciana, por todo amor, carinho, compreensão e principalmente paciência.

A Deus, pela oportunidade de realização de mais este importante passo em minha vida.

Ao BankBoston, pelo apoio e suporte no desenvolvimento deste trabalho.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A estrutura original da arquitetura de protocolos que define a rede de computadores Internet, conhecida pela sigla TCP/IP[1], propõe um ambiente de rede aberto com poucos mecanismos de segurança nativos. Agrega-se a isto a natureza pública das referências (RFC - *Request For Comments*) e implementações padrões do pacote de protocolos TCP/IP.

De fato, a natureza pública e aberta, aliada à falta de mecanismos de proteção, tornaram o aspecto de segurança um dos principais problemas estruturais da arquitetura TCP/IP. Se, por um lado, estes aspectos originais foram os principais responsáveis pelo sucesso e popularização da Internet, a deficiência no aspecto de segurança representa um dos maiores limitadores ao uso desta tecnologia, em especial às aplicações voltadas ao mundo comercial.

Visto em um ambiente aberto e distribuído, o conceito de segurança deve ser bastante amplo. O espectro varia desde a segurança da aplicação e do sistema operacional, até o canal de comunicação propriamente dito. Embora esses aspectos sejam de igual relevância no contexto geral da Internet, o presente trabalho tem por objetivo cobrir apenas os problemas relativos ao canal de comunicação. Ainda assim, o referido aspecto pode ser analisado sob dois enfoques: controle de acesso de pacotes e protocolos de segurança, definidos através de técnicas de criptografia.

Ao se conectar uma rede local à Internet, pode-se aumentar a segurança dos sistemas locais reduzindo a conectividade externa a determinadas máquinas e serviços específicos. Os elementos que conferem este controle, através de técnicas de filtragem e seleção de pacotes sobre as próprias definições dos protocolos, são conhecidos como *firewalls*. Os *firewalls* foram, a grosso

1. Referência aos principais protocolos da Arquitetura: TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*).

modo, os primeiros elementos estruturais introduzidos na arquitetura TCP/IP com intuito específico de agregar segurança à Internet.

Entretanto, conceitos mais avançados como autenticação de usuário e servidor, privacidade e integridade de dados, só podem ser obtidos através de técnicas sofisticadas de criptografia. A demanda por estes serviços, por parte das aplicações e protocolos básicos de infra-estrutura, induz à definição de protocolos de segurança. Estes são atrelados a estas técnicas de forma a implementar as premissas de segurança requeridas.

Devido à sua complexidade e estudo ainda recente, em comparação às técnicas de segurança baseadas em *firewalls*, os protocolos de segurança representam o escopo principal deste trabalho de dissertação.

Atualmente o leitor pode facilmente encontrar literatura sobre a arquitetura TCP/IP. No entanto, o mesmo não acontece para o modelo da arquitetura TCP/IP segura. Apesar de diversas referências tratarem de aspectos de segurança isoladamente, tais como vulnerabilidades dos protocolos, *firewalls*, e criptografia, pouco existe a respeito dos protocolos seguros em si além das referidas RFCs. Além disso, o leitor carece de uma visão global de como obter segurança no ambiente global da Internet.

Desta forma, o presente trabalho visa cobrir esta lacuna de revisão bibliográfica da questão de segurança na arquitetura TCP/IP, cobrindo portanto os aspectos de vulnerabilidades, controle de acesso (*firewalls*), criptografia, protocolos e infra-estrutura de segurança. O objetivo intrínseco é a obtenção de canais seguros de comunicação na Internet. Em acréscimo ao aspecto prático de criptografia voltado para especialistas em rede, a classificação dos níveis dos protocolos de segurança objetiva prover uma visão ampla de aplicabilidade das diversas técnicas e serviços de segurança, além de sugestões de uso e melhoramento.

Em resumo, o Capítulo 2 apresenta os problemas de segurança da estrutura original dos protocolos TCP/IP e o capítulo seguinte apresenta as técnicas convencionais de segurança por controle de acesso e *firewalls*. O Capítulo 4 apresenta de maneira informal, ou seja, sem rigor matemático, os conceitos de criptografia. Já o Capítulo 5 trata do problema de gerenciamento e distribuição de chaves em ambientes abertos de maneira genérica.

Os principais protocolos de segurança propostos para a suíte TCP/IP são apresentados no Capítulo 6. Naturalmente, não se objetiva uma descrição quantitativa de todos os protocolos propostos, mas sim uma amostra qualitativa dos mesmos. Os protocolos são então classificados e

abordados conforme sua camada de implementação, desde o nível de redes até o de aplicação. Assim, esse capítulo retrata a arquitetura IP segura (RFC1825), seguida pelos mecanismos de sessão abordados no Capítulo 7. No Capítulo 8, referente à camada de aplicação, são abordadas aplicações de comércio eletrônico, correio eletrônico seguro e protocolos estruturais tais como o SSH (*Secure Shell)* e o DNSSEC (DNS Seguro). Após as conclusões obtidas neste trabalho, o Apêndice A resume os principais algoritmos de criptografia, respectivamente simétricos e assimétricos, e o Apêndice B descreve uma aplicação dos conhecimentos tratados neste trabalho.

Capítulo 2

Vulnerabilidades dos protocolos TCP/IP

Neste capítulo serão abordadas as vulnerabilidades inerentes à arquitetura TCP/IP. Por "inerentes" entende-se aqui a concepção original ou implementações padrão. É necessário lembrar que a suíte de protocolos TCP/IP evolui continuamente, e que parte das vulnerabilidades originais têm sido corrigidas recentemente, ao passo que outras têm requerido modificações profundas nos protocolos básicos. O leitor é referido à Seção 6.9, página 85, que trata brevemente da nova versão do protocolo IP, o IPv6.

2.1 Introdução

Uma das principais deficiências no aspecto de segurança do protocolo IP é a incapacidade deste de autenticar uma máquina na rede. Em outras palavras, com base no endereço IP de origem de um pacote recebido, é impossível determinar com certeza a identidade da máquina que o tenha originado. Há também poucas garantias de que o conteúdo de um pacote recebido não tenha sido alterado, muito menos ainda que a privacidade dos dados nele contidos tenha sido preservada.

Os ataques que exploram tal falha têm como tática mais comum a personificação de uma máquina na rede. Sua finalidade é a de obter informações sigilosas como senhas, abusar da confiança que máquinas mantêm entre si, até ações mais sutis e sofisticadas, como alterar o conteúdo dos dados que estejam de passagem para outros destinos.

O enfoque aqui abordado será analisar o impacto dessas vulnerabilidades na segurança de transações baseadas no protocolo TCP/IP, bem como o papel delas na prevenção do problema. Para uma visão mais específica das deficiências e soluções adotadas para cada protocolo da

arquitetura TCP/IP, o leitor interessado deve consultar o clássico artigo de Steven Bellovin sobre o assunto [14]. Porém, antes de sugerir soluções, é necessário compreender um pouco mais a fundo os procedimentos que possibilitam personificar uma máquina na rede, e a partir daí, as formas de prevenção.

2.2 Vulnerabilidades do meio físico

As tecnologias de redes locais apresentam certas vulnerabilidades inerentes à sua própria natureza. Ataques que exploram estas vulnerabilidades são normalmente restritos às próprias redes locais, mas também podem ser usados, como descrito a seguir, para sobrepujar protocolos de mais alto nível na hierarquia TCP/IP.

As deficiências da tecnologia Ethernet, que constitui a maior parte das redes locais, expõem ainda mais as fragilidades da Internet. Os principais problemas estão relacionados com:

- facilidade de se realizar grampo (*eavesdropping*)
- falso mapeamento entre endereço de rede (IP) e endereço físico (ARP).

2.2.1 Grampeando a rede

Sendo Ethernet uma tecnologia de rede onde o meio físico é compartilhado, é possível configurar a interface de rede de uma máquina em modo "promíscuo", e assim receber todos os quadros transmitidos no meio. Em sistemas UNIX esta facilidade está disponível somente ao super-usuário, mas em sistemas operacionais como DOS, Windows e MacOS não existe obviamente nenhuma restrição de acesso à interface de rede.

Geralmente tem-se por objetivo obter informações privilegiadas, como por exemplo senhas, mas também pode-se usar tal facilidade como passo na implementação de ataques mais sofisticados, tal qual "sequestro de sessão", que será tratado na Seção 2.7, página 10.

2.2.2 Falso mapeamento entre endereço IP e endereço Ethernet (ARP)

O protocolo ARP (*Address Resolution Protocol*) mapeia um endereço IP em endereço Ethernet enviando uma mensagem de *broadcast*[1] perguntando qual o endereço Ethernet da interface que possui o endereço IP conhecido. A máquina que tiver o endereço IP procurado, ou alguma outra

agindo em nome daquela, responde com o par: endereço IP - endereço Ethernet. Uma máquina mal intencionada pode então enviar respostas falsas, desviando todo o tráfego para si, tendo como objetivo personificar uma máquina, ou mais sutilmente, modificar os dados que estiverem sendo transmitidos entre duas outras máquinas. *firewalls* e redes locais fisicamente inseguras devem mapear endereços físicos de maneira estática. De qualquer forma pouco pode ser feito em relação a conexões externas que passem por redes Ethernet inseguras.

2.3 Criando rotas para ataque

A dificuldade em se personificar completamente uma máquina está em fazer com que pacotes enviados à máquina legítima cheguem ao falso destino. Para que isto seja possível, os roteadores entre o alvo do ataque e o atacante devem de alguma forma ser subvertidos. Isto pode ser efetuado de duas maneiras:

- utilizando a opção *source routing* [29] do protocolo IP — pacotes IP que contenham tal opção são por especificação roteados não com base nas tabelas de rota dos roteadores, mas conforme determinado *a priori* pelo originador do mesmo
- alterando as tabelas de rotas dos roteadores envolvidos — os protocolos de propagação de rotas (notadamente RIP) tipicamente não têm como averiguar as informações recebidas, sendo que os roteadores acreditam piamente nestas informações; isto torna possível alterar todas as tabelas de rotas entre uma máquina alvo (atacada) e uma atacante, de maneira a personificar uma terceira, na qual a atacada confia

Na seção seguinte será mostrado que as mensagens de redirecionamento ICMP *redirect messages* [29] [83], podem ser exploradas com o mesmo intuito que os protocolos de propagação de rotas.

2.4 ICMP (*Internet Control Message Protocol*)

O protocolo ICMP pode ser usado para implementar ataques de negação de serviço através de mensagens *destination unreachable* ou por *address mask replies* falsos [29] [83]. Para amenizar o efeito deste tipo de ataque, mensagens ICMP deveriam sempre estar atreladas a uma conexão

1. Difusão; transmissão de pacote pelo endereço de difusão que todas as máquinas devem receber ("ouvir").

específica [23]. Tendo em vista tal objetivo, os primeiros 8 bytes do cabeçalho da camada de transporte (UDP/TCP) são sempre incluídos nas mensagens ICMP. No caso de implementações mais modernas de TCP/IP, como por exemplo no Solaris da Sun Microsystems, são 64 bytes, valor definido através da variável *icmp_return_data_bytes*. Contudo, implementações ICMP antigas não fazem uso desta informação, e todas as conexões entre duas máquinas são então afetadas. Claramente, estas implementações são extremamente vulneráveis a ataques de negação de serviço, que serão futuramente melhor abordados.

Mensagens geradas por roteadores escrutinadores (dispondo de filtro de pacotes), em conseqüência de pacotes enviados a portas bloqueadas, podem "derrubar" todas as conexões com uma determinada máquina. A geração de mensagens *destination unreachable* por parte de um roteador deve ser feita de maneira a não prejudicar conexões legítimas.

No entanto, o perigo maior reside na possibilidade de se injetar rotas falsas através de ICMP *redirect messages*. O que torna este tipo de ataque mais difícil de ser implementado é que este tipo de mensagem só pode ser gerado em função de conexões existentes, e deve ter sido originado de um *gateway* conectado na mesma sub-rede da máquina a ser atacada.

Em geral, todas as implementações do protocolo ICMP realizam averiguações antes de alterar as tabelas de roteamento. Uma máquina 4.4BSD, por exemplo, averigua as seguintes informações [83]:

- o novo roteador indicado pela mensagem deve estar em uma rede diretamente conectada
- o *redirect* deve ter sido gerado pelo roteador atual do destino a ser alterado
- a rota a ser modificada deve ser uma rota indireta
- *redirect messages* somente podem alterar rotas para máquinas, e não rotas para redes

Considerando apenas o fator segurança, roteadores e máquinas que implementam *firewalls* não deveriam atualizar suas tabelas de roteamento com base em *redirect messages*. Este tipo de mensagem, quando originada de redes externas, caracteriza problemas de configuração ou tentativa de ataque. A melhor defesa neste caso é filtrar tais mensagens através de um *firewall*.

2.5 UDP (*User Datagram Protocol*)

O protocolo UDP não mantém nenhuma informação (estado) a respeito das "sessões" por ele realizadas. Em outras palavras, não há procedimento de estabelecimento de conexão nem números seqüências, como no protocolo TCP. Isto torna o UDP muito suscetível à falsificação de endereços IP (IP *spoofing*).

Entretanto, por questões de desempenho, o protocolo *Remote Procedure Call* (RPC) da Sun Microsystems [72] foi implementado sobre UDP. Serviços de extrema importância e sensibilidade do ponto de vista de segurança, como NFS e NIS [85], tiveram como base o RPC, e portanto padecem das mesmas limitações inerentes ao protocolo UDP[1]. Apesar de também apresentar problemas [55], o protocolo RPC Seguro é de longe mais confiável que o RPC convencional e seu uso, sempre que possível, deve ser encorajado.

2.6 *Sequence Number Attacks* (SNA)

Protocolos que mantém estado de suas conexões[2] ou que realizam *queries*[3] por intermédio de números seqüenciais são, em geral, menos vulneráveis a IP *spoofing* que protocolos que não mantém estado [29]. Esta segurança só existe, no entanto, caso os números seqüenciais usados nas transações não possam ser previstos. É portanto de fundamental importância que tais números sejam gerados da maneira o mais aleatória possível.

A maior parte dos geradores de números aleatórios utilizam realimentação de suas saídas. Na verdade, os geradores de seqüências "aleatórias", que muitas vezes nem chegam a ser implementados (como no caso de muitos *resolvers* que sempre iniciam suas *queries* com um identificador igual a zero [83]), são facilmente previsíveis na maioria dos protocolos[4]. Isto torna possível a um atacante, como será descrito para o caso específico do protocolo TCP, prever a próxima seqüência a ser utilizada por uma máquina e, desta forma, sobrepujar a "proteção" oferecida pelos números seqüenciais.

---

1. Afora as debilidades do protocolo UDP, os protocolos RPC, NFS e NIS apresentam vulnerabilidades próprias que fogem do escopo deste trabalho.
2. Por exemplo o protocolo TCP - *Transmission Control Protocol.*
3. Perguntas realizadas a servidores; como por exemplo, ocorre nos protocolos DNS - *Domain Name System* e EGP - *Exterior Gateway Protocol.*
4. A maioria dos fabricantes tem atualizado seus softwares, e apenas uma minoria permanece trivialmente previsível.

2.6.1 TCP *Sequence Number Prediction Attack* (SNPA)

TCP SNPA nada mais é que um caso específico de ataques SNA. No entanto, devido à importância do protocolo TCP e às peculiaridades envolvidas, este tipo de ataque será abordado em maiores detalhes.

Apesar de já ter sido previsto por Steve Bellovin [14] em 1989, somente após o natal de 1994, quando Kevin Mitnick usou desta técnica para penetrar nos computadores de Mitomu Shimomura, que este ataque se tornou famoso, passando então a não mais ser considerado uma proeza difícil de se implementar.

SNPA é baseado na capacidade de se prever o número seqüencial inicial, a ser usado pela máquina alvo, no processo de estabelecimento de uma conexão TCP, conhecido como "triplo aperto de mão" [29] (*three way handshake*). Suponhamos que um determinado intruso INT tenha previsto o próximo número seqüencial inicial NSI a ser usado por um servidor SRV. O atacante poderia então se passar por uma máquina confiável CON, emitindo comandos "r" de Berkeley, por exemplo, sem ter que se autenticar. O ataque se desenvolveria da seguinte forma (ver figura abaixo):

1. INT envia um pacote: SYN(NSI_{INT}) para SRV, com endereço IP de origem de CON
2. SRV responde com SYN(NSI_{SRV}), ACK(NSI_{INT}) para CON
3. INT nunca recebe a resposta acima (seria necessário desvirtuar roteadores, como descrito na seção acima), mas como INT "adivinhou" NSI_{SRV}, INT consegue estabelecer conexão enviando ACK(NSI_{SRV}) para SRV
4. dados maliciosos podem então ser enviados de INT para SRV; o interessante é que INT nem mesmo precisa receber os pacotes enviados por SRV, pois o ataque a SRV pode ser feito às cegas[1].

NSI são passíveis de previsão, já que na maioria dos sistemas estes são derivados do último número seqüencial usado. Então, a partir de uma conexão legítima é possível calcular o próximo número seqüencial a ser usado por uma máquina, desde que novas conexões não se realizem.

1. INT pode, por exemplo, fazer o equivalente a `rsh SRV echo + > /.rhosts`.

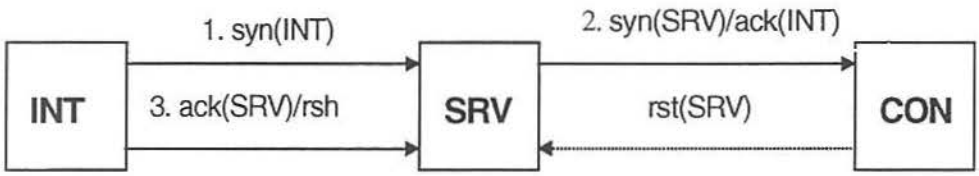

Figura 2.1: TCP *Prediction Number Attack.*

Não foi assim "por acaso" que o computador de Shimomura tenha sido invadido em uma noite de natal. Com acesso "exclusivo" a uma máquina, um atacante pode efetuar inúmeras conexões sucessivas até acertar um NSI definido por esta.

Apenas uma questão não foi abordada na descrição deste ataque: ao receber a confirmação da tentativa de estabelecimento de conexão vinda de SRV, que na verdade foi ilegalmente efetuada por INT, CON normalmente tentaria "derrubar" a conexão fraudulenta, enviando um pacote de *reset* (com a opção RST habilitada) para SRV. Contudo, CON pode estar inoperante, inclusive por obra de ataques de negação de serviço, que serão tratados adiante.

2.7 Sequestro de sessão (*Session Hijacking* - SHA)

Mesmo que se tenha em mente *one time passwords*[1], não é possível evitar totalmente o acesso não autorizado a uma rede privada pela Internet. Para um atacante, que esteja estrategicamente posicionado entre um usuário legítimo fora da rede privada e o sistema alvo, existe a possibilidade de se poder tomar controle da sessão do usuário com o sistema, após este ter sido devidamente autenticado. Devido à restrição de "posicionamento" do atacante, este ataque também é classificado como uma espécie de *man_in_the_middle attack.*

Ao atacante é requerido a habilidade de acompanhar a sessão de autenticação de um usuário legítimo, possivelmente grampeando uma rede Ethernet que esteja entre ambos, e de interromper em tempo hábil a comunicação entre o usuário e o sistema, logo após este ter sido corretamente autenticado. Isto naturalmente pode ser obtido, negando-se serviço à máquina do usuário. Se for possível ao atacante, pelos métodos anteriormente descritos, alterar os dados enviados pelo usuário, não há sequer necessidade de se implementar negação de serviço.

1. Método de autenticação baseado em desafio e resposta, onde senhas reutilizáveis (senhas convencionais) nunca são transmitidas pela rede.

Ainda que não sirva como consolo, é fundamental estabelecer que um ataque SHA é relativamente difícil de ser implementado devido às circunstâncias apresentadas anteriormente. Apesar de *one time passwords* estarem um patamar acima em termos de segurança que senhas convencionais, o risco de se ter usuários acessando uma rede local via sistemas externos não deve ser desprezado no contexto atual da Internet.

2.8 Negação de serviço e "ataques de baixo nível"

2.8.1 Fragmentação IP

Para operar entre redes locais heterogêneas, um pacote IP pode ser fragmentado toda vez que exceder o limite do maior quadro que uma determinada rede local é capaz de transmitir (MTU—*Maximum Transfer Unit*) [29]. Neste caso é necessário dividir o pacote IP em fragmentos menores que a MTU.

Quando um datagrama IP é recebido, o valor do campo FO (*fragment offset*) e o estado do bit MF (*more fragment*) ligado permitirão descobrir se o datagrama IP original sofreu fragmentação ou não; em caso positivo, a recepção irá armazenar temporariamente os fragmentos para permitir a reconstrução do datagrama. Os campos endereço de origem e destino, protocolo e ID (*Identification*) identificam fragmentos de um mesmo pacote IP [29]. O campo FO (*fragment offset*), por sua vez, define a ordem dos fragmentos no pacote IP. A RFC 791 [69], descreve um algoritmo de remontagem de fragmentos que assume a sobreposição de fragmentos, caso o valor do campo FO seja inferior ao tamanho do fragmento anterior. Esta característica do protocolo IP pode ser usada para atacar *firewalls* baseados em filtros de pacotes e implementar ataques de negação de serviço como veremos a seguir.

2.8.1.1 Ataque a filtros de pacotes

Como o cabeçalho dos protocolos de nível superior como TCP e UDP encontram-se no primeiro fragmento IP (*offset* = 0), alguma informação deve ser mantida caso se queira filtrar os demais fragmentos. Normalmente não é possível remontar o pacote IP original sem o primeiro fragmento, mas deixar os demais fragmentos passarem é um risco que deve ser evitado.

Com relação aos fragmentos IP, mais duas questões são interessantes de serem citadas. O primeiro fragmento (com *offset* = 0) pode chegar ao destino depois que algum outro já tenha chegado. Uma solução não ideal, porém segura, seria então descartar este fragmento [71] [60].

O outro problema refere-se a fragmentos que sejam menores que 68 octetos. Como o cabeçalho IP pode ter até 60 octetos, 8 octetos de dados não são suficientes para transportar todo o cabeçalho do protocolo TCP (em 8 octetos é possível ter apenas os seguintes campos: porta de origem e destino e número seqüencial). Informações importantes como os *flags* TCP (*code bits*) não têm, neste caso, como ser consideradas no processo de filtragem.

Um atacante pode também tirar proveito das regras de remontagem de pacotes IP para sobrepujar as regras de filtragem de um *firewall*. Para tanto, um fragmento IP com *offset* = "1" (8 octetos) pode ser usado para sobrepor (alterar) informações importantes no processo de filtragem do cabeçalho TCP, tal como, os *flags* TCP citados acima.

Para prevenir os ataques relativos a fragmentação IP, um *firewall* deve filtrar todos fragmentos que possuam o campo FO igual a "1" e cujo protocolo de transporte seja o protocolo TCP [73].

2.8.2 Negação de serviço (*Denial of Service Attacks*)

Ataques de negação de serviço têm se tornado cada vez mais frequentes na Internet. Normalmente eles exploram o limite da capacidade de sistemas de processar requisições de serviços e mensagens (ou pacotes), através de técnicas de saturação ou *flooding*. Falhas na implantação de serviços, como por exemplo estouro de pilha (*buffer overruns*), e nas camadas de baixo nível da arquitetura TCP/IP, também são exploradas neste tipo de ataque.

2.8.2.1 Ataques a aplicações

Novamente, a essência dos ataques a aplicações se baseia no princípio de saturar a capacidade de processamento dos serviços e falhas no tratamento de *buffers* e pilhas. Alguns exemplos são bombardeio de mensagens de correio eletrônico (*mail bombing*), ataques aos serviços `chargen` e `daytime` em sistemas UNIX, dentre outros.

2.8.2.2 *Flooding Attacks*

Máquinas na Internet estão sujeitas a ataques de baixo nível que exploram deficiências nas implementações das camadas mais baixas do TCP/IP, em especial os protocolos IP e TCP.

Outros protocolos, como o UDP e o ICMP, podem também sofrer ataques específicos. Serviços baseados no protocolo UDP podem ser desabilitados por saturação de pacotes UDP (UDP *flooding*). O mesmo princípio pode ser usado, através do protocolo ICMP, para indisponibilizar uma máquina e saturar o tráfego de uma rede local.

Ao receber uma mensagem ICMP do tipo *echo request* outra mensagem do tipo *echo reply* é retornada para a máquina origem pela máquina destino[1]. Portanto, uma mensagem de *echo request* pode ser enviada para o endereço de uma sub-rede (por exemplo o endereço 143.106.0.0), tendo o endereço de origem alterado para uma determinada máquina alvo. Neste caso, todas as máquinas da sub-rede irão responder com uma mensagem do tipo *echo reply* para a máquina alvo, inundando a máquina e rede alvos. Este ataque é popularmente conhecido como Smurf [44].

Uma variação mais perigosa deste ataque faz do endereço de origem como sendo o próprio endereço de *broadcast*, fazendo um único pacote do atacante multiplicar-se *ad infinitum*, saturando a rede atacada e sem causar tráfego entre o atacante e o alvo.

2.8.2.3 Ataques ao IP

Algumas implementações do algoritmo de desfragmentação IP não tratam corretamente fragmentos IP sobrepostos, como descrito na seção anterior. Mais ainda, pacotes "indevidamente" ou particularmente construídos, como por exemplo pacotes TCP com a *flag* SYN ligada e endereço de origem e destino iguais, podem "derrubar" determinadas implementações do protocolo TCP. Ferramentas de domínio público, conhecidas respectivamente como "Teardrop" e "Land" [27] exploram as vulnerabilidades descritas acima. Notadamente, algumas combinações de *flags* "não permitidas" do protocolo TCP, como por exemplo SYN/FIN, e variações da construção descrita acima podem também derrubar uma máquina.

2.8.2.4 SYN *Flooding*

O ataque conhecido como SYN *Flooding* é, no entanto, um dos exemplos mais interessantes de ataques de negação de serviço na arquitetura TCP/IP. No processo de estabelecimento de uma conexão TCP [29] uma máquina, ao receber um pacote TCP com a *flag* SYN ligada, deve respon-

1. O comando Ping utiliza esta funcionalidade do protocolo ICMP.

der com um pacote SYN/ACK para a máquina que iniciou a conexão. Somente após receber um segundo pacote ACK a conexão TCP é finalmente estabelecida.

Caso a máquina originadora do pacote SYN não envie o segundo pacote ACK, a conexão fica em estado pendente no *cache* de conexões semi-abertas do *kernel* da máquina destinatária. Através deste artifício é possível estourar o *cache* de conexões semi-abertas de uma máquina, evitando-se que outras conexões possam ser estabelecidas.

É importante observar que o endereço IP de origem da máquina atacante pode ser estabelecido de maneira aleatória. Este fato torna bastante difícil monitorar e identificar a fonte do ataque. Procedimentos independentes da especificação TCP podem porém ser definidos para contornar este problema. Uma possível solução seria identificar conexões TCP pendentes "antigas" e fechá-las através de pacotes RST enviados à máquina alvo. Contudo, é conceitualmente impossível determinar uma solução definitiva para este problema.

2.8.2.5 *Stealth Scanning*

Internet *scanners*, como são chamadas as ferramentas de auditoria de rede, têm por objetivo procurar por serviços e falhas que possam comprometer uma máquina. A maior parte destas ferramentas baseia-se no estabelecimento de conexões TCP com todas as portas de uma máquina, de maneira a determinar se estas estão ativas ou não. Felizmente, TCP *wrapper* [86] e outros programas afins, chamados *scanner detectors*, possuem a capacidade de contabilizar todas as conexões efetuadas, e a partir daí determinar a ação de tais *scanners*.

Por outro lado, não é necessário estabelecer uma conexão TCP para determinar se uma porta está ativa ou não. Enviando-se segmentos TCP em várias combinações de bits de *flags* (SYN, ACK, FIN, RST, URG, PUSH), e sendo conhecida a resposta a ser devolvida por um determinado sistema, tem-se o mesmo efeito ou pior do que estabelecer uma conexão por completo.

De [51] foi extraída uma tabela com segmentos enviados e respostas recebidas[1] que exemplifica o que fora dito no parágrafo anterior:

1. As resposta variam de acordo com a implementação (plataforma em questão). A Referência [71] apresenta uma descrição completa dos vários pacotes possíveis e correspondentes respostas para vários tipos de sistemas.

flag	resposta porta ativa	resposta porta inativa
SYN	SYN/ACK	RST ou nenhuma resposta
SYN/FIN	ACK ou SYN/ACK	RST
ACK	nenhuma resposta	RST

Tabela 2.1: Tabela de resposta a pacotes TCP com *flags* fora de especificação.

2.8.3 Defesas

Este último ataque talvez seja um dos mais difíceis de se evitar por completo. É normalmente complicado até mesmo determinar com certeza que um ataque do gênero esteja ocorrendo ou tenha ocorrido. A geração de *logs*[1] e análise periódica dos dados armazenados, através de ferramentas como *swatch* [41], é ainda a melhor forma de se lidar com este tipo de problema.

Prevenções mais específicas normalmente referem-se à defesa da máquina em si (*defense in depth*) [40] e ferem o escopo deste trabalho. Mesmo assim, seguem abaixo algumas considerações para prevenir tais ataques:

- definir partições de disco especiais para diretórios ou arquivos que possam receber grande volume de dados, como por exemplo arquivos de *logs* e diretório PUB de FTP e *mail*
- procurar evitar a instalação de serviços públicos (FTP anônimo, servidor HTTP etc) e desnecessários
- limitar, quando disponível no sistema o número máximo de processos sendo executados e o acesso destes aos recursos do sistema, como o uso de CPU e de memória
- procedimentos de ajuste de desempenho (*performance tuning*), além de uma prática recomendada em geral, têm efeito significativo na disponibilidade do sistema, que na verdade é também um fator de medida de segurança
- atualizar sistema operacional e aplicativos com correções dos fabricantes para vulnerabilidades descobertas, uma vez que quase todas as vulnerabilidades dos protocolos TCP/IP de baixo nível têm sido corrigidas pelos fabricantes de cada plataforma.

1. Registros de eventos.

2.9 DNS (*Domain Name System*)

Como DNS é um banco de dados distribuído, existe pouco controle sobre a veracidade das informações divulgadas pelos inúmeros domínios existentes na Internet. É possível, por exemplo, que um atacante que controle o mapeamento inverso de algum domínio, tenha algum endereço IP local (do atacante) sendo mapeado no nome de uma máquina confiável de um outro sistema qualquer (sistema alvo). Desta forma, este atacante pode se passar pela máquina confiável se a autenticação no sistema alvo for baseada no nome de quem estabeleceu a conexão, como no caso dos comandos "r" de Berkeley [15].

Tal tipo de ataque pode, e de fato é, evitado na maioria dos sistemas, comparando-se o mapeamento inverso com o mapeamento direto (*cross-check*) em chamadas de sistema como *gethostbyname*. Mesmo assim, autenticação baseada em nomes é ainda mais fraca que a baseada no endereçamento IP. Isto porque um ataque ao DNS pode ser mais facilmente implementado que ataques baseados em IP *spoofing*. Por este motivo, sempre que possível, arquivos de configuração de sistema devem conter os endereços IP das máquinas e não os seus nomes.

À parte o problema de autenticação, o DNS oferece em geral informações preciosas (para um atacante) a respeito de uma rede privada. Uma boa política é a de não autorizar transferências entre servidores secundários (*zone transfer*) [87], mas isto pode não ser suficiente para um atacante determinado e paciente (eles geralmente o são). Este pode então "varrer" todo o espaço de endereçamento via pedidos de revolução inversas.

Um enfoque mais "paranóico", proposto por Brent Chapman e descrito detalhadamente em [30], é ter um servidor de nomes oferecendo o mínimo possível de informação ao mundo externo e um outro servidor interno para uso doméstico. De forma bem resumida tem-se a seguinte situação: as aplicações no servidor externo seriam configuradas (através do arquivo /etc/resolv.conf) para enviar *queries* ao servidor interno. Este por sua vez, propagaria ao servidor externo (opção *forward query*) questões relacionadas com os domínios externos. Desta maneira é possível isolar completamente as informações de nomes internos e inclusive manter na rede privada endereçamento IP não oficial.

Capítulo 3

Internet *firewalls*

Neste capítulo serão abordadas as principais técnicas usadas em *firewalls*, abordando topologias, componentes básicos e técnicas empregadas na construção de tais sistemas, junto com breves análises de segurança de cada variante apresentada.

3.1 Introdução

Para proteger uma rede privada contra as ameaças externas oriundas de uma conexão com a Internet, dois enfoques são possíveis: "reforçar" a segurança dos sistemas de rede e serviços em geral; e isolar a rede interna, restringindo o acesso externo através de um *firewall*.

Como já fora indiretamente mencionado, o enfoque em um estudo sobre *firewalls* é a defesa de uma rede privada em relação ao mundo externo (Internet). *Firewalls* por si oferecem pouca ou nenhuma proteção contra ataques oriundos de usuários da própria rede privada. Estes, portanto, oferecem defesa ao perímetro de segurança, mas não necessariamente incrementam o nível de proteção individual de cada máquina localmente. Tampouco agregam segurança ao canal de comunicação propriamente dito. Preocupações com a proteção de cada máquina em si constituem um tópico de extrema importância, mas que fogem do escopo do assunto *firewalls*.

3.2 Conceitos básicos

Firewalls são um conjunto de sistemas (uma ou mais máquinas/roteadores) situado entre uma rede privada e a Internet, que têm como principal função interceptar todo tráfego entre ambos, e com base na política de segurança interna, permitir ou não a sua passagem [91].

Conforme o nível onde atuam na arquitetura em camadas do TCP/IP, os elementos de um *firewalls* são classificados em filtros de pacotes (*firewalls* nos níveis de rede e de transporte) e *Application Gateways* (*firewalls* no nível de aplicação). Filtros de pacotes (*screening routers*) podem ser implementados em *hardware* dedicado (roteadores) ou através de *software* rodando em máquinas de uso geral: têm capacidade de efetuar filtragem de pacotes com base nas informações contidas até o nível de rede, tais como endereço IP de origem e destino, protocolo de transporte (TCP/UDP), portas de origem e destino dos protocolos de transporte etc.

Application Gateways ou *proxies* são programas específicos, implementados para cada aplicação que se queria permitir passagem através do *firewall*. Pelo fato de atuarem na camada de aplicação, os *application gateways* oferecem normalmente maior controle de acesso que os filtros de pacotes. Vantagens e desvantagens do uso de cada um ou ambos os modelos acima serão analisadas na seção seguinte, bem como em um estudo simplificado das várias combinações de topologias possíveis.

3.3 Componentes e topologias básicas

Na prática, uma série de combinações de configurações são possíveis na construção de um *firewall*. *Application level gateways* podem ser combinados com filtros de pacotes (*screening routers*) produzindo diversas topologias. Abaixo serão apresentados os principais elementos e topologias encontrados.

3.3.1 Componentes básicos de um *firewall*

3.3.1.1 Filtro de pacotes

Pode ser tanto um roteador comercial (p. ex. CISCO), como uma máquina especialmente configurada para tal. A filtragem é efetuada com base nos seguintes critérios: protocolo (UDP, TCP etc), endereço IP e portas (caso TCP ou UDP) de origem e destino, além dos campos de controle específicos de cada protocolo. Muitos *firewalls* consistem apenas de um roteador com capacidade de efetuar filtragem.

3.3.1.2 *Bastion*[1] *host*

Diz-se de uma máquina configurada para atender um determinado serviço de rede, potencialmente vulnerável a ataques devido à complexidade do serviço ou de sua implementação. O termo "configurada" significa que a máquina deve ser especialmente fortificada, i.e. minimizada em potenciais fragilidades. Pelo fato de o serviço ser provido à Internet em geral, a máquina fica mais exposta a ataques, e daí a necessidade de fortificação especial, exatamente como nos *bastions* medievais.

3.3.1.3 *Dual-homed host*

Em redes baseadas em TCP/IP *multi-homed host* descreve uma máquina que tem múltiplas interfaces de rede, isto é, está conectada a mais de uma rede física (mais de uma interface Ethernet, p. ex.).

Um *dual-homed host* é um tipo especial de *multi-homed host* que tem duas interfaces de rede com a função de roteamento desabilitada, ou seja, com a habilidade de propagar pacotes IP (IP *forwarding*) desativada. Um *dual-homed host* pode ser usado para implementar um *firewall* sem um roteador propriamente dito. Por definição um *dual-homed host* é um *bastion host*.

3.3.1.4 *Packet state filter (adaptive screener ou dynamic filter)*

Packet state filters (PSF) podem também ser implementados tanto em um roteador comercial como por um *software* especializado em um *bastion host*. Um PSF mantém o estado das conexões estabelecidas, e desta forma, filtra os pacotes em função das regras estáticas pré-estabelecidas (política de segurança) e também pelo contexto das conexões estabelecidas (permitidas). Esta característica, acrescida do conceito híbrido de uma única máquina implementando filtragem de pacotes em suas interfaces de rede e *proxies* específicos, define a idéia de *firewalls* de terceira geração.

1. parte altamente fortificada de um castelo medieval

3.3.2 Topologias

3.3.2.1 Roteador escrutinador

Muitas redes são protegidas apenas por roteadores com capacidade de filtrar pacotes. Neste caso existe comunicação direta entre as várias máquinas na rede privada e a Internet. Assim a zona de risco (máquinas que podem ser conectadas da rede externa) é igual ao número de máquinas na rede[1]. A preservação da integridade da rede, neste caso, depende da "correta" configuração e administração de todas as máquinas internas.

No caso de invasão do *firewall*, torna-se muito difícil detectar tal fato, especialmente se for usado roteador sem capacidade de gerar *logs*. No caso dos roteadores serem erroneamente configurados ou apresentarem algum ponto de falha, a rede interna fica totalmente desprotegida.

Este tipo de configuração é uma das poucas topologias de *firewalls* que implementam a filosofia: "o que não é expressamente proibido é permitido". A fragilidade da topologia, no entanto, proporciona alto grau de liberdade no acesso à Internet.

3.3.2.2 *Dual-homed gateway*

Como já fora mencionado, este tipo de *gateway* não propaga tráfego TCP/IP entre suas interfaces diretamente. O *gateway* age como uma barreira entre a Internet e a rede local. O acesso aos serviços externos pode ser implementado de duas formas: dando aos usuários permissão de conectar no *gateway* ou através de *proxies* (neste caso também conhecido como servidor *proxy*). A primeira forma também implementa uma política de "o que não é expressamente proibido é permitido" e o acesso de usuários ao *gateway* o torna muito vulnerável.

A segunda configuração é normalmente a mais utilizada. Neste caso tem-se uma política de "o que não é expressamente permitido é proibido". Em funcionamento normal a zona de risco se restringe ao *gateway*. O modo de falha, isto é, a capacidade de se detectar que o *firewall* foi comprometido é o aspecto mais frágil desta configuração. Se o servidor *proxy* (*gateway*) é destruído, é possível que um oponente habilidoso reabilite o roteamento e deixe toda a rede aberta para ataques.

---

1. Normalmente, porém, é costume limitar quais máquinas internas podem aceitar quais serviços, através do filtro.

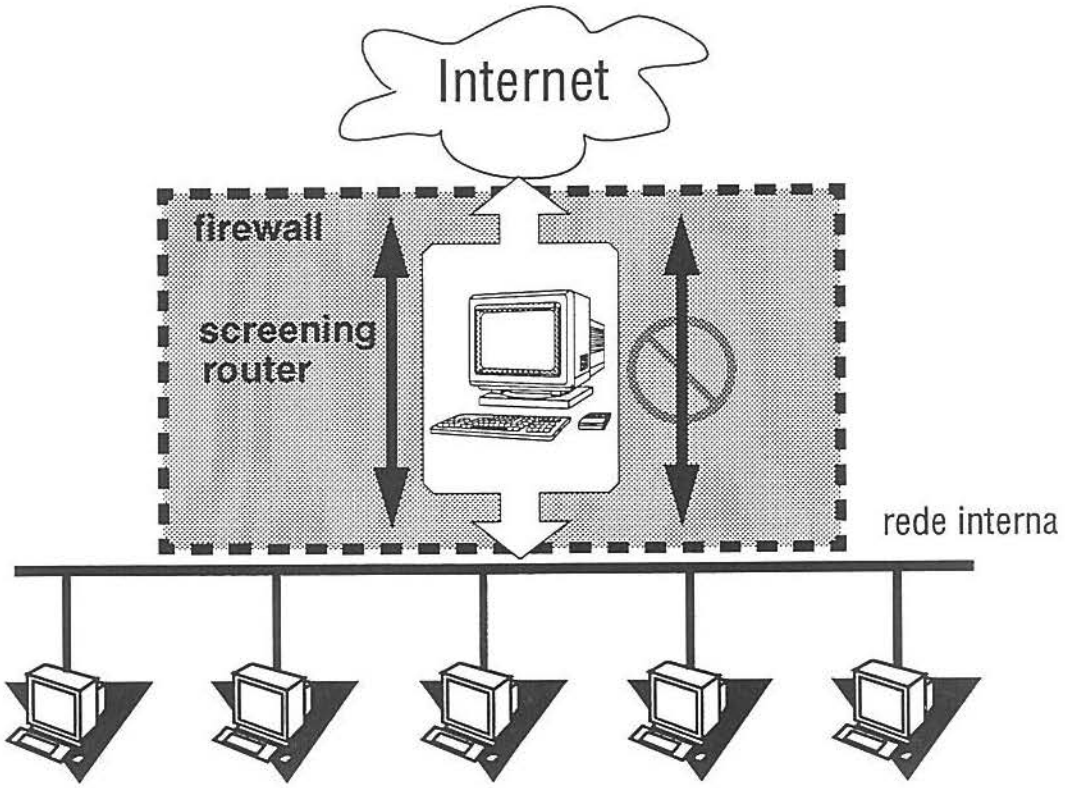

Figura 3.1: Rede protegida apenas por roteador com filtragem (roteador escrutinador simples).

3.3.2.3 *Screened gateway*

Geralmente esta topologia é bastante segura e relativamente fácil de se configurar. Tipicamente, um *bastion host* é configurado na rede privada com um roteador entre esta e a Internet. Somente tráfego externo para o *bastion host* é permitido passar pelo roteador.

A zona de risco é restrita ao *bastion host* e ao roteador. Se um atacante invadir o *bastion host*, existe uma razoável quantidade de opções para atacar a rede interna. Em muitos aspectos, este enfoque é similar à topologia *dual-homed gateway*. Desta forma, ambas apresentam considerações de implementação e modos de falhas semelhantes.

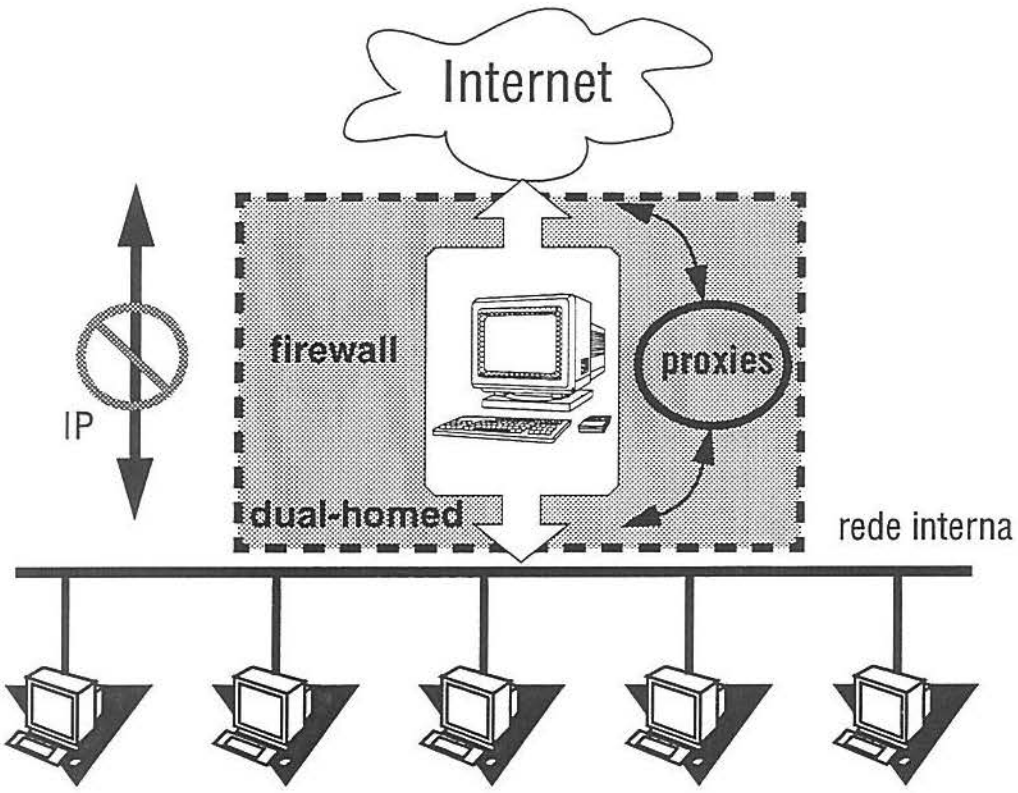

Figura 3.2: *Dual-homed gateway* (servidor *proxy*).

3.3.2.4 *Screened subnet*

Nesta topologia aparece a figura de uma sub-rede entre a rede local e a Internet (Figura 3.4, página 24), também conhecida como zona desmilitarizada (DMZ - *Demilitarized zone*). Não é permitido tráfego direto entre a rede local e a Internet. Ambas podem apenas acessar a sub-rede, normalmente através de um *bastion host* entre esta e a rede privada. Existe um filtro de pacotes entre a Internet e a sub-rede, e outro entre esta e a rede privada. O primeiro tem como função impedir o tráfego direto entre a rede privada e a Internet. Já o segundo tem por objetivo proteger a rede privada em caso de comprometimento do *bastion host*. Isto pode ser feito limitando o acesso deste à rede local.

O conceito de zona desmilitarizada também pode ser obtido através de *firewalls* de terceira geração (Figura 3.5, página 25) que agreguem a capacidade de filtrar pacotes e *proxies* específicos em um único *box*. Tais elementos podem tanto ser implementados puramente em *software* (*multi-homed host*) ou através de roteadores comerciais.

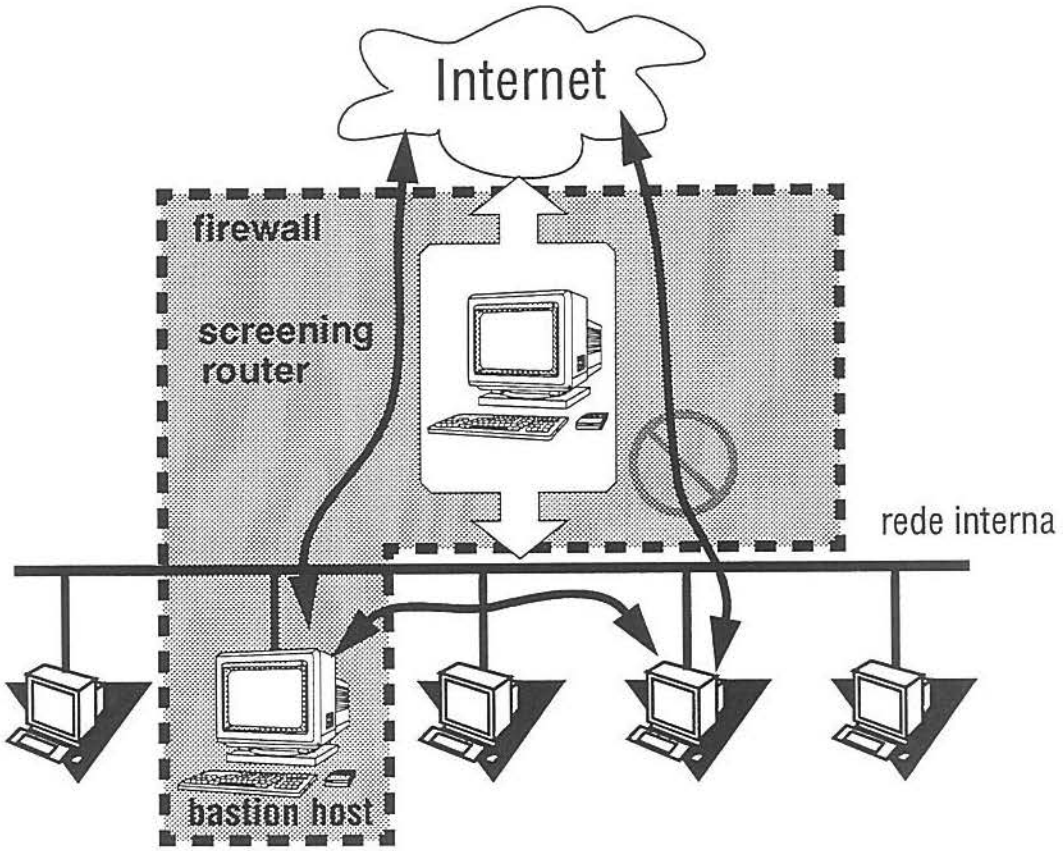

Figura 3.3: *Screened gateway* (roteador escrutinador com *gateway*).

A zona de risco é pequena: *bastion host* e filtros de pacotes. Os serviços são implementados através de *proxies* e por isto esta topologia representa uma política de uso do tipo: "o que não é expressamente permitido é proibido".

Para destruir o *firewall,* um oponente deve reconfigurar o roteamento nos dois roteadores sem se deixar perceber. Fato possível, mas que pode ser dificultado, desabilitando o acesso aos roteadores (apenas por console), ou restringindo o mesmo a algumas máquinas específicas da rede privada. Neste caso, um atacante teria que invadir o *bastion host*, e daí, uma das máquinas internas, e finalmente o roteador. Tudo isto sem deixar ser percebido.

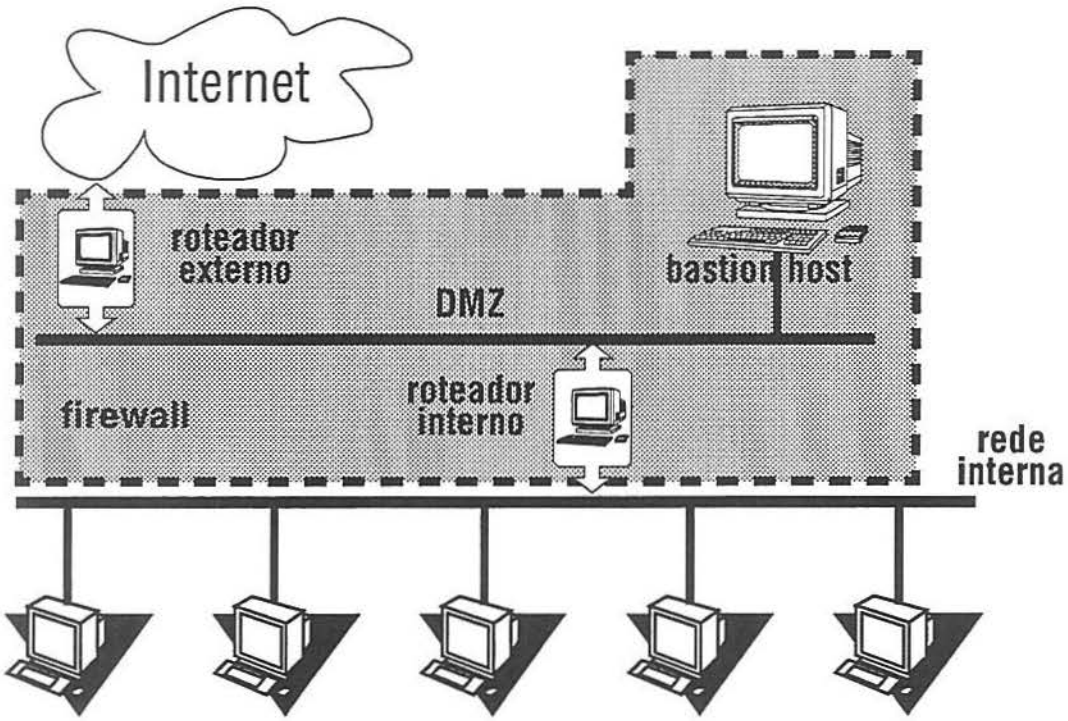

Figura 3.4: *Screened Subnet* (zona desmilitarizada com *bastion host*).

3.4 Filtro de pacotes

A título de revisão, filtragem de pacotes é a capacidade que alguns roteadores (roteadores propriamente ditos ou *gateways)* têm de rejeitar pacotes com base nas seguintes informações: protocolo, endereço IP de origem e destino, portas e campos de controle.

Em geral, nenhum contexto é mantido, e a filtragem é baseada unicamente no conteúdo do pacote em questão. De acordo com o roteador, a filtragem pode ser efetuada na chegada ou na saída dos pacotes. A filtragem na saída é implementada em alguns roteadores por questão de desempenho, uma vez que os pacotes podem ser tratados de uma só vez, quando da consulta à tabela de rotas. No entanto, do ponto de vista de segurança, a filtragem na chegada é preferida, pois desta forma é possível configurar a interface externa do roteador para rejeitar pacotes que contenham o endereço IP de origem correspondente ao endereço de alguma rede interna. Com isso, evita-se a falsificação de endereços IP (*IP spoofing*), tornando mais confiável a autenticação de máquinas na rede interna com base no endereçamento IP.

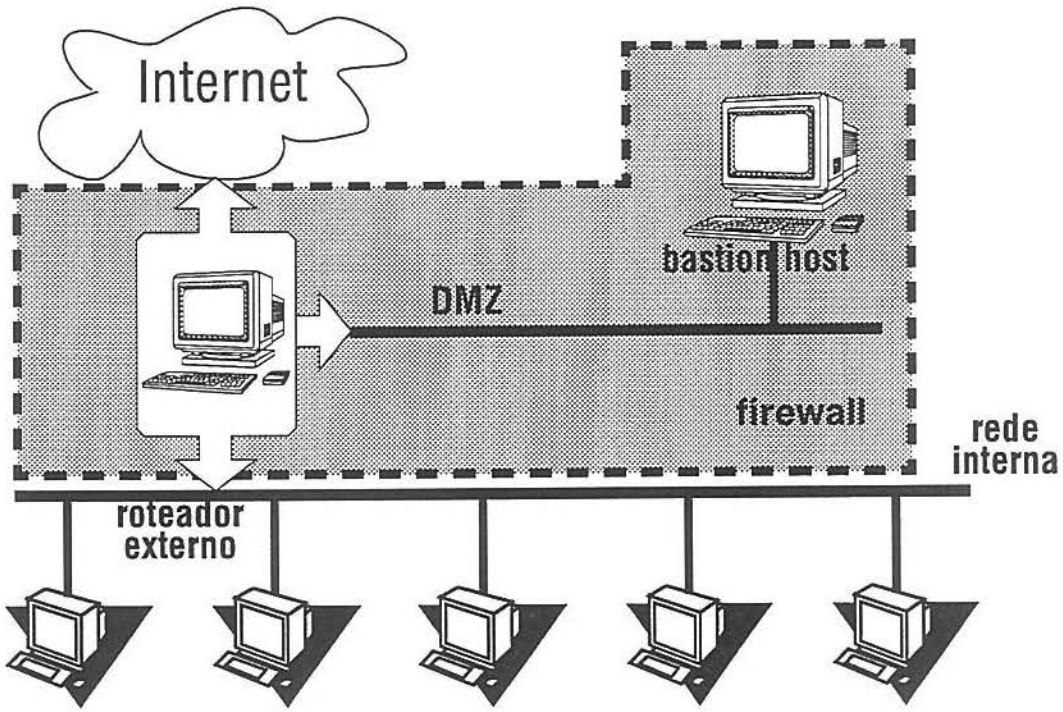

Figura 3.5: DMZ em um *firewall* de terceira geração.

Às vezes, é interessante que se possa estabelecer conexões em apenas um sentido. Especificamente, na topologia *screened subnet* é possível proteger a rede interna em caso do *bastion host* ser comprometido, configurando o segundo roteador para permitir que apenas as máquinas internas estabeleçam comunicação com o *bastion host* e não no sentido inverso.

No caso de conexões TCP pode-se definir o sentido da comunicação com base no bit de reconhecimento e no de sincronismo (ACK/SYN bit).

A tabela abaixo [23] exemplifica um conjunto de regras de filtragem que implementam uma política de permitir conexões externas somente ao serviço de correio eletrônico e livre acesso das máquinas internas à Internet via protocolo TCP. Note que apenas as conexões TCP estabelecidas (ACK bit) têm acesso no sentido Internet para a rede local.

Ação	Origem	Porta	Destino	Porta	Flag	Comentários
Permitir	*	*	GW	25		*mail relay*
Permitir	Rede Local	*	*	*		Acesso Interno à Internet

Tabela 3.1: Estrutura tabular das regras de filtragem convencionais.

Ação	Origem	Porta	Destino	Porta	Flag	Comentários
Permitir	*	*	Rede Local	*	ACK	respostas chamadas internas
Bloquear	GW	*	Rede Local	*		impedir GW conectar R. L.

Tabela 3.1: Estrutura tabular das regras de filtragem convencionais.

No caso de "sessões" (em oposição a "conexões") baseadas no protocolo UDP, a filtragem deve ser baseada somente na porta e endereços IP de origem e destino, uma vez que sem a abstração de conexão não é possível determinar o sentido da "sessão".

3.5 *Application gateways*

Application level gateways ou *proxies* são programas específicos que permitem a propagação de uma aplicação (Telnet, FTP etc). Como estes são programados para atuar em nível de aplicação podem, contrariamente a um filtro de pacotes, controlar o acesso e manter "logs" do uso das aplicações.

Novamente, por serem implementados em nível de aplicação e extremamente especializados para cada tipo de serviço, *proxies* propiciam elevado nível de controle de acesso, inclusive no nível de usuários e não apenas de máquinas. O serviço FTP pode, por exemplo, ser restrito à transferência de arquivos apenas no sentido de fora para dentro (*inbound*), restringindo-se a execução do comando `put/mput` pelo agente *proxy*. Naturalmente é possível dar permissão para que apenas alguns usuários possam transferir arquivos para fora da rede Interna, permitindo acesso aos mesmos ao comando FTP `put/mput`.

Proteção anti-vírus, para arquivos transferidos via os protocolos FTP e SNMP (correio eletrônico), somente pode ser obtida através de programas específicos instalados nos respectivos *proxies* de serviços. Outros serviços como *real-audio*, X11, dentre outros, exigem também o uso de *proxies* para se obter elevado nível de controle de acesso e segurança.

A principal desvantagem de um *proxy* é que para cada serviço oferecido deve haver um programa específico. Portanto, apenas um número limitado de serviços podem ser oferecidos desta forma através de um *firewall*. Além disto, o acesso nem sempre é totalmente transparente do ponto de vista do cliente. Primeiro este deve se conectar ao *proxy* para depois acessar o serviço externo requerido.

3.6 Packet State Filters (PSF)

Filtros de pacotes tomam suas decisões baseados unicamente nos pacotes recém-chegados. Nenhuma informação a respeito dos pacotes anteriormente tratados é levada em consideração no processo decisório vigente. Em outras palavras, nenhuma informação a respeito dos pacotes aceitos ou rejeitados é mantida.

A desvantagem deste enfoque pode ser novamente ilustrada pelo protocolo FTP[1]. O modo de operação normal do protocolo requer que conexões TCP sejam estabelecidas de fora para dentro da rede interna, mesmo para transferência de arquivos no sentido inverso. Isto porque o protocolo FTP através do comando PORT especifica que o servidor conectado deve estabelecer uma conexão TCP, para transferir um arquivo requisitado, com o cliente na porta definida por este no próprio comando PORT.

Para implementar a política de acesso definida na Seção 3.4, página 24, de permitir acesso a todas as máquinas internas à Internet, a regra abaixo (Tabela 3.2, página 27) deveria ser incluída na Tabela 3.1, página 25, para o serviço de FTP. Todos os serviços suportados em portas acima de 1024 ficarão sujeitos a ataques externos. Naturalmente todo este problema poderia ser evitado caso os clientes FTP fizessem uso do comando PASV[2] [23].

Ação	Origem	Porta	Destino	Porta	Flag	Comentários
Permitir	*	*	*	>1024		permitir clientes FTP

Tabela 3.2: Permitir que os clientes FTP funcionem em modo padrão.

Por manterem informações de estado e terem acesso a módulos *proxies* específicos, os PSFs podem tratar o protocolo FTP padrão de maneira transparente e segura. Ao registrar um comando PORT por parte de um cliente interno, o *firewall* permitirá apenas o estabelecimento de conexões externas (fora para dentro) oriundas do servidor FTP previamente conectado e somente nas portas especificadas pelo comando PORT, dentro de uma janela de tempo pré-determinada (*time-out*).

---

1. O protocolo FTP é particularmente de grande interesse didático no estudo dos *firewalls*, devido às suas características de duplo uso de portas, sentido de conexão invertido (do servidor para o cliente), e primitivas de protocolos razoavelmente simples.

2. Neste caso, ao contrário do comando PORT, o cliente estabelece conexões para transferência de dados nas portas especificadas pelo servidor.

Mesmo para protocolos *stateless* como o UDP é possível, por intermédio de PSFs, adicionar alguma noção de estado. A idéia é que, quando se envia um pacote UDP para fora, uma resposta é esperada dentro de um certo período de tempo no sentido oposto. Os PSFs podem, com o princípio de *time-out,* dar a idéia de conexão e direção no protocolo UDP, mesmo que ainda de forma um tanto rudimentar se comparado ao TCP. Mesmo assim qualquer informação sobre o estado de uma "conexão" contribui para torna-lá mais segura.

Os PSFs são também muito eficientes para evitar ataques de baixo nível na arquitetura TCP/IP. No protocolo TCP, por exemplo, a decisão de se rejeitar ou aceitar uma conexão é efetuada por filtros de pacotes sobre os pacotes que possuem o bit de sincronismo ligado (SYN *flag*). Os pacotes que não possuem tal bit ligado são incapazes de estabelecer conexão com qualquer serviço ativo, sendo neste sentido inofensivos. No entanto, estes pacotes podem ser usados, como visto na Seção 2.8.2.5, página 14, na procura por serviços ativos - *stealth scanning*.

Um *packet state filter* (PSF) que acompanhe os números seqüenciais (*ack/sequence*) de uma conexão em andamento, pode evitar *stealth scanning* rejeitando os pacotes cujos números seqüenciais não estejam dentro da janela TCP correta (falso ACK, por exemplo), ou identificar pacotes SYN que sejam ataques de *stealth scanning* ou SYN *flooding* (ver Seção 2.8.2.4, página 13) por não serem seguidos de pacotes ACK correspondentes.

Por último, os PSFs podem levantar listas do fluxo de pacotes gerados, e com isso oferecer uma análise de desempenho em baixo nível. Excessos de segmentos RST podem também ser detectados através dos log gerados e assim produzir pistas sobre o mau andamento das conexões, e possivelmente sobre tentativas de TCP SNPA (ver Seção 2.6.1, página 9) ou *stealth scanning*.

3.7 Adicionando criptografia aos *firewalls*

Soluções mais efetivas para os ataques que exploram as vulnerabilidades do protocolo TCP/IP certamente envolvem o uso de alguma técnica de criptografia. Infelizmente limitações técnicas e comerciais (restrição de exportação de *software* que usem criptografia) tendem a limitar o uso extensivo de criptografia.

Enquanto não se tem uma solução de cunho genérico (se é que tal solução realmente existe) talvez o melhor que se possa ter sejam "feudos seguros" defendidos por *firewalls*. Novamente tem-se a questão de garantir acesso seguro a usuários que estejam fora dos limites de proteção,

especialmente àqueles que estejam além dos limites físicos de uma rede privada e portanto suscetíveis a sequestro de sessão (computação nômade).

Há dois enfoques possíveis no uso de criptografia em *firewalls* [92]:

- criptografia de *firewall* para *firewall*
- criptografia de usuário (externo) ao *firewall*

3.7.1 Criptografia de *firewall* para *firewall*

Desta maneira é possível estabelecer comunicação confiável entre dois *firewalls* quaisquer na Internet. O que se tem na verdade é a definição de uma rede privada virtual (ou VPN, *Virtual Perimeter Network*), ou seja a partir de duas redes locais separadas fisicamente, pode-se criar uma única rede virtual usando a estrutura de comunicação existente na Internet.

A comunicação entre *firewalls* pode ser implementada em vários níveis do protocolo TCP/IP. Quando implementada no nível de aplicação, apenas a parte de dados do protocolo TCP é cifrada. Pode ser interessante que o TCP *checksum* seja calculado a partir do texto original antes de ser cifrado. Assim, pacotes adulterados podem ser detectados antes que cheguem à camada superior, onde seria necessário implementar alguma forma de retransmissão para tratar tal problema. Por outro lado, se totalmente implementado no nível de aplicação, não há necessidade de se alterar o protocolo TCP original, o que pode tornar a implementação mais fácil, especialmente quando não se tem acesso aos códigos fonte da plataforma sendo usada.

A forma intermediária é ter todo o segmento TCP, incluindo o cabeçalho, cifrado. Neste caso, uma camada responsável por cifrar e decifrar o conteúdo dos dados do protocolo IP deve ser definida logo acima deste. Retransmissões e *checksums* são processados normalmente pelo protocolo TCP, possibilitando o uso de aplicações existentes de modo transparente.

Contudo, o mais versátil e seguro é fazer tunelamento IP entre *firewalls,* inclusive como forma de prevenir análise de tráfego entre os segmentos de rede. Um pacote IP a ser enviado de uma rede local para outra, ao passar pelo primeiro *firewall*, seria cifrado e encapsulado em outro pacote IP, que passaria então a ter como endereço de origem e destino os endereços IP dos *firewalls* envolvidos. No segundo *firewall* o pacote seria desencapsulado e posteriormente enviado ao destino final já na rede local remota. Mais uma vez é possível ter endereçamento IP não oficial, mas desta vez em todo o perímetro de uma rede virtual e não apenas em uma rede fisicamente

isolada. Este esquema é normalmente implementado nos *firewalls* comerciais que dispõem de recursos de criptografia.

Este conceito será nova e mais detalhadamente abordado no Capítulo 6, que trata da arquitetura IP segura.

3.7.2 Criptografia de usuário para *firewall*

Mesmo com o conceito de rede privada virtual, o problema de se ter um usuário fora do perímetro de proteção continua existindo. A melhor forma de garantir o acesso seguro de usuários externos, além do uso de autenticação forte (*one time passwords* ou algum protocolo que faça uso de criptografia, por exemplo) é ter toda a sessão, entre o usuário distante e o sistema, cifrada.

A idéia é evitar personificação por autenticação forte, e sequestro de sessão usando criptografia. Mesmo que um atacante tenha sucesso em um ataque do tipo sequestro de sessão, os pacotes enviados por este seriam decifrados em lixo e a sessão seria então interrompida. Supõe-se naturalmente que o atacante não seja capaz de "quebrar" o sistema de criptografia usado, o que normalmente é bem menos trivial que simplesmente explorar as falhas nativas do TCP/IP.

Neste caso, um possível enfoque seria implementar criptografia no nível de aplicação. Algum *application gateway* teria que ser responsável por tornar os serviços acessíveis aos usuários de maneira transparente pelo *firewall*.

A primeira dificuldade vislumbrada é com relação à necessidade de se ter aplicações cliente especialmente desenvolvidas para tal uso. É sem dúvida um grande inconveniente não poder fazer uso das várias aplicações clientes já existentes na Internet. A falta de padronização atual tende a levar a sistemas proprietários, quase sempre totalmente incompatíveis entre si.

Novamente, a implementação de protocolos seguros no nível de aplicação será amplamente abordada no Capítulo 8. Por último, vale notar que poucos *firewalls* comerciais tratam completamente o problema de garantir acesso seguro a transações para usuários que estejam na Internet (acesso externo).

Capítulo 4

Introdução à criptografia computacional

O propósito deste capítulo é introduzir os principais conceitos de criptografia ao leitor, sem contudo o rigor matemático próprio à área. Os principais protocolos utilizados ou de possível utilidade à Internet serão cobertos.

4.1 Conceitos de segurança

Infelizmente, sem o uso de criptografia não é possível garantir segurança à informação que transita pela Internet, especialmente quando os canais de comunicação usados são inseguros. Mesmo que o meio de comunicação seja inviolável, conceitos como integridade e autenticação são características essenciais para segurança de muitas aplicações.

Genericamente, os seguintes requisitos são necessários à implementação de aplicações seguras na Internet:

- **Autenticação de origem** – capacidade de verificar se a identidade proclamada em mensagens recebidas é de fato verdadeira.
- **Integridade** – indicação inequívoca de que as mensagens transmitidas entre dois nós (emissor e receptor) não tenham sido adulteradas acidentalmente ou por terceiros ao longo do trajeto entre estes nós. Isto é, caso uma mensagem seja alterada em trânsito, tal alteração pode ser detectada pelo receptor da mesma.

- **Privacidade** – garantia de que apenas as partes envolvidas diretamente na comunicação, isto é apenas o emissor e receptor, sejam capazes de determinar o conteúdo dos dados que estão sendo transmitidos.
- **Não-repúdio** - capacidade do receptor de provar que alguém de fato lhe enviou uma determinada mensagem, independentemente da disposição do emissor em aceitar ou não tal fato.
- **Propagação perfeita de segredo** - propriedade que determinados sistemas possuem de manter a privacidade das mensagens anteriormente transmitidas, mesmo que a segurança do sistema tenha sido comprometida; ou seja, apenas as mensagens presentes e futuras podem ser comprometidas, mas não as passadas.

A implementação de tais propriedades implicam necessariamente no uso de técnicas de criptografia, às quais este capítulo fará uma breve introdução. Sugerimos ao leitor interessado em maiores detalhes consultar uma das referências existentes sobre o assunto [93] [79].

4.2 Sistemas de criptografia

Algoritmos de ciframento são funções matemáticas usadas para tornar ilegível um conjunto (bloco) de dados de entrada, para quem não tenha acesso à chave de deciframento. A rigor, um par de chaves e funções são usados para cifrar e decifrar os dados em questão.

Notação

Suponhamos que uma função de ciframento $\boldsymbol{E}$ seja usada com uma chave $\boldsymbol{K}$ para transformar um texto claro $\boldsymbol{P}$ em um texto cifrado $\boldsymbol{C}$. Então escrevemos:

$$C = E_K [P]$$

O deciframento é efetuado pela função inversa $\boldsymbol{D}$ e chave de deciframento $\boldsymbol{K}^{-1}$:

$$P = D_{K^{-1}} [C]$$

4.2.1 Sistemas simétricos ou de chave secreta

Os sistemas de criptografia simétricos são caracterizados pelo fato da chave de ciframento ser idêntica à chave de deciframento ($\mathbf{K} = \mathbf{K}^{-1}$). Estes sistemas podem ser divididos em duas categorias: *stream ciphers* e *block ciphers*.

Os primeiros operam bit a bit sobre o fluxo de dados (*stream*) de entrada, combinando-o com outro fluxo de dados que constitui a chave simétrica (normalmente uma seqüência pseudo-aleatória derivada de uma chave de tamanho fixo). Este algoritmos são também conhecidos como **cifradores bit a bit**[1] ou **de caracteres** (*stream ciphers*). Já os **algoritmos cifradores de bloco** (*block ciphers*) atuam sobre um bloco de dados de tamanho fixo.

Os algoritmos simétricos modernos são, na maioria dos caso, embaralhadores de bits derivados a partir de operações matemáticas básicas e funções lógicas tipo Ou-Exclusivo, deslocamento de bits, substituições e permutações simples. Apesar de, teoricamente, ser possível implementar um algoritmo através de operações mais elaboradas, o enfoque acima é mais eficiente em termos de desempenho e implementação em *hardware* e *software*.

O objetivo dos algoritmos simétricos conhecidos é obter os princípios de ***confusão*** e ***difusão***. O princípio de confusão obscurece a relação entre o texto claro e o cifrado, e é normalmente obtido através de técnicas de substituição. Já a difusão elimina a redundância do texto claro, espalhando-a sobre o texto cifrado. O mecanismo mais usado para implementar difusão é através de permutações e também através de substituições.

Os algoritmos cifradores de caracteres foram bastante usados antes da existência de computadores. Atualmente, em sistemas de criptografia computacionais, cifradores de blocos de 64 bits são os mais comumente utilizados. Este tamanho de bloco é considerado grande o suficiente para garantir a segurança do algoritmo, e pequeno o bastante para os computadores atuais operarem. Contudo, devido ao avanço nas técnicas de criptoanálise e a presença de arquiteturas computacionais de 64 bits, cifradores de blocos de 128 bits têm sido propostos. O algoritmo AES (*Advanced Encryption Standard*), a ser proposto pelo NIST[2] em substituição ao padrão DES, requer operação sobre blocos de 128 bits [1].

Ainda hoje, o algoritmo cifrador de blocos mais conhecido e usado é o DES (*Data Encryption Standard*). Devido à sua importância para o desenvolvimento da criptografia moderna, o DES é também a referência clássica no assunto.

---

1. O termo **cifrador bit a bit** nos parece mais apropriado para descrever o conceito de *stream ciphers*. No entanto, por motivos de praticidade e sonoridade, o termo **cifrador de caracteres** é usado doravante neste documento.
2. Instituto de padrões do governo dos Estados Unidos da América.

4.2.1.1 DES (*Data Encryption Standard*)

A estrutura do algoritmo DES é baseada no conceito de **rede de Feistel**. Após uma permutação inicial **IP**, o bloco de dados é dividido em dois blocos de 32 bits (R_0 e L_0). Sobre estes são efetuados 16 ciclos de operações idênticas (iterações), combinados com 16 sub-chaves derivadas da chave secreta ***K***, conforme a expressão abaixo:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i), \text{ para } 1 \leq i \leq 15$$

No último estágio (**i = *16***), os blocos não são permutados, ou seja:

$$L_{16} = L_{15} \oplus f(R_{15}, K_{16}) \text{ e } R_{16} = R_{15}$$

Em seguida, os dois blocos de 32 bits são novamente reagrupados e uma permutação final $\mathbf{IP}^{-1}$, inversa da inicial, define o conteúdo final do bloco cifrado. A mesma estrutura é usada no processo de decifrамento, exceto que a ordem de geração das sub-chaves é invertida.

Vale salientar que a função ***f*** pode ser tão sofisticada quanto se queira. De fato, este é o bloco fundamental de ciframento do DES, que é constituído de uma combinação de duas técnicas básicas de ciframento: substituição e permutação. Ou seja, uma permutação/expansão, seguida de uma substituição baseada na chave secreta **K**, denominada *S-Box*. Finalmente, mais uma permutação denominada *P-Box*, define a saída da função ***f***. É na substituição *S-Box*, não linear, que reside toda a força do algoritmo em si. A figura abaixo ilustra uma iteração do algoritmo DES descrita acima.

As chaves $\mathbf{K_i}$ são geradas através de operações de deslocamento de bits e permutações sobre a chave original **K**. A chave secreta **K** tem um comprimento de 64 bits, sendo que cada oitavo bit é simplesmente usado para verificação de paridade. O tamanho real da chave de ciframento é conseqüentemente de apenas 56 bits.

4.2.1.2 Outros algoritmos simétricos

O Apêndice I apresenta uma lista dos algoritmos disponíveis ao público em geral. O objetivo é oferecer uma referência rápida sobre a segurança e disponibilidade dos mesmos, sem no entanto se ater a detalhes técnicos (matemáticos) específicos. Para um estudo mais técnico o leitor interessado deve referir-se a [93].

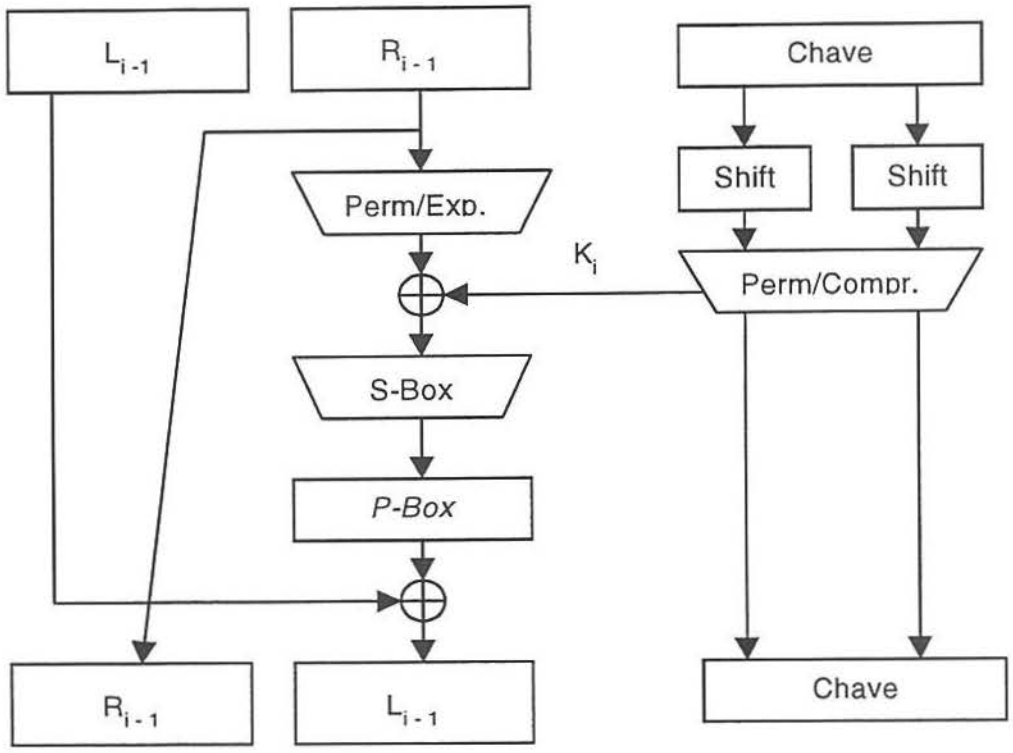

Figura 4.1: Iteração do algoritmo DES.

4.2.1.3 Modos de operação

Um sistema de criptografia pode operar um mesmo algoritmo de ciframento de diversas maneiras distintas. Existem quatro modos básicos de operação, ou **transformações**, a saber:

Electronic code book

Modo básico de operação: a mensagem é cifrada pela função **E** sem nenhuma operação adicional. Ou seja,

$$C_n = E_k[P_n] \text{ e } P_n = D_k[C_n]$$

onde Pn é o n-ésimo bloco.

Conseqüentemente, toda vez que blocos iguais forem cifrados com a mesma chave, o valor resultante será o mesmo. Se um oponente for capaz de efetuar ataques de **texto claro conhecido** (ver próxima seção), um catálogo de blocos cifrados e seus respectivos significados pode ser facilmente construído. Tal catálogo pode, devido à estrutura repetitiva e redundante das men-

sagens em geral, ser usado para decifrar mensagens que tiverem sido cifradas com a chave geradora do catálogo em questão.

Além disso, ataques de **recorte e colagem** e **repetição de mensagens** (Seção 4.3, página 39) são também factíveis, já que os blocos de uma mensagem podem ser removidos, repetidos, ou substituídos com intuito de modificar o significado da mesma. Em outras palavras, sistemas que operam unicamente neste modo não garantem **integridade** às mensagens cifradas.

Cipher Block Chaining

O CBC é o modo de operação mais usado em **transformações** que operam sobre protocolos de comunicação. Neste modo, para cada bloco de texto claro a ser cifrado é efetuada a operação lógica de OU-Exclusivo com o bloco cifrado anteriormente, antes do algoritmo de ciframento em si. Isto é,

$$C_n = E_k [P_n \oplus C_{n-1}]$$

A operação inversa de deciframento é

$$P_n = C_{n-1} \oplus D_k[C_n]$$

O bloco inicial $\mathbf{C_0}$, conhecido como vetor de inicialização (IV - *Initialization Vector*) deve ser transmitido ou acordado entre as partes envolvidas. Para que mensagens idênticas, cifradas com a mesma chave, gerem textos distintos, evitando assim os ataques descritos para o modo ECB, o **IV** deve ser único para cada mensagem transmitida. No entanto, este não precisa ser secreto, podendo ser transmitido em claro.

Algum método para identificar o último bloco transmitido também deve ser estabelecido a *priori*. Cada bloco cifrado depende não somente do bloco de texto claro que o gerou, mas também de todos os blocos cifrados anteriormente. Se um bloco cifrado for corrompido, o bloco imediatamente subseqüente a este também será perdido. Entretanto, a partir deste ponto os demais blocos são decifrados corretamente. Por isto, o modo CBC é considerado **auto regenerativo**.

Output Feedback Mode

O modo OFB é uma maneira de transformar um cifrador de blocos em um cifrador de caracter síncrono. O OFB usa o cifrador de blocos simplesmente como um gerador de seqüência aleató-

ria, realimentando sua entrada com sua saída, e efetuando a operação de OU-Exclusivo com o texto claro para produzir o seguinte texto cifrado:

$$C_n = P_n \oplus S_n;\ S_n = E_k [S_{n-1}]$$

A operação inversa usa a mesma seqüência aleatória (S_n):

$$P_n = C_n \oplus S_n;\ S_n = E_k [S_{n-1}]$$

A seqüência $\mathbf{S_n}$ pode, na verdade, ser dividida em octetos em um registrador de deslocamento e conseqüentemente ser usada para cifrar/decifrar caracteres deste mesmo tamanho, conforme ilustra a Figura 4.2, página 37, para um cifrador de caracteres de 8 bits.

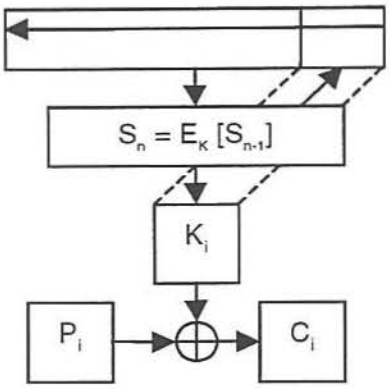

Figura 4.2: *Output feedback mode.*

A seqüência $\mathbf{S_n}$ é completamente independente do texto claro e cifrado em questão. Assim, boa parte da operação de ciframento/deciframento pode ser feita antes da existência do texto claro, ou da chegada do texto cifrado. No entanto, o estado inicial de $\mathbf{S_0}$ deve ser estabelecido através de um vetor de inicialização **IV** único, não necessariamente secreto.

O OFB apresenta a propriedade de não propagar erros. Um erro em um bit do texto cifrado provoca exatamente um bit de erro no texto claro decifrado. Por outro lado, a operação de OU-Exclusivo do OFB oferece oportunidade para um oponente, que tenha habilidade de interceptar e alterar mensagens, de introduzir erros específicos no texto claro de maneira **imperceptível** a cada caracter transmitido.

Cipher Feedback Mode

O modo CFB segue as mesmas características do modo anterior, exceto que a seqüência aleatória $\mathbf{S_n}$ é definida pelo bloco cifrado anterior $\mathbf{C_{n-1}}$. Isto é:

$$C_n = P_n \oplus S_n;\ S_n = E_k\,[C_{n-1}]$$

e a operação inversa:

$$P_n = C_n \oplus S_n;\ S_n = E_k\,[C_{n-1}]$$

Neste caso, a seqüência aleatória $\mathbf{S_n}$ não pode ser calculada previamente. Novamente, a seqüência $\mathbf{S_n}$ pode, na verdade, ser dividida em octetos em um registrador de deslocamento e conseqüentemente ser usada para cifrar/decifrar caracteres deste mesmo tamanho, usando os bits menos significativos da seqüência aleatória na operação de OU-Exclusivo com o texto claro (ver figura abaixo).

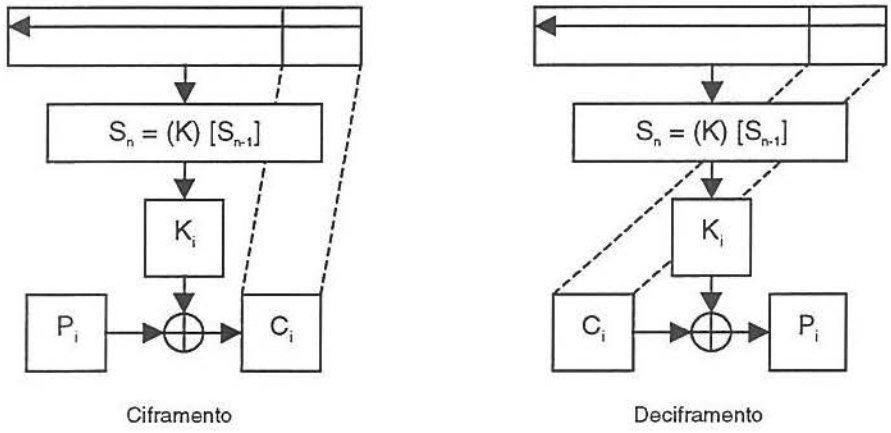

Figura 4.3: *Cipher-feedback mode.*

Um único bit de erro no texto cifrado provoca um erro no bit de texto claro correspondente e em todo bloco de texto claro subseqüente. Assim erros podem ser propositalmente introduzidos em bits específicos do texto claro decifrado, ao custo de deteriorar todo o bloco seguinte.

4.3 Tipos de ataques

Infelizmente, até mesmo os sistemas de criptografia mais sofisticados podem, eventualmente, ser comprometidos. De fato, existe uma grande variedade de ataques aos quais os mesmos estão sujeitos. Para analisar a pior situação, normalmente pressupõe-se que o oponente tenha pleno acesso ao meio de comunicação e conhecimento do sistema de criptografia a ser atacado.

O objetivo desta seção será introduzir o leitor aos tipos básicos de ataques, bem como à terminologia usada para descrevê-los.

4.3.1 Ataques de força bruta

Este ataque consiste em tentar todas as chaves possíveis para decifrar uma determinada mensagem. Neste caso, a segurança do algoritmo de criptografia depende do tamanho da chave implementada.

4.3.2 Criptoanálise

Criptoanálise é a ciência de decifrar um texto sem ter acesso direto à chave de deciframento/ciframento. O objetivo da criptoanálise é obter, a partir de um texto cifrado, o texto claro original ou até mesmo a chave usada para cifrá-lo. Normalmente é explorada uma fraqueza matemática do algoritmo e modo de operação em uso, ou do protocolo de comunicação.

De acordo com o acesso que um oponente tenha sobre o par texto cifrado/texto claro em um ataque, estes são classificados em quatro categorias básicas:

4.3.2.1 Texto cifrado puro

Este ataque é o mais difícil do ponto de vista da criptoanálise. Nele o oponente tem acesso somente à várias mensagens cifradas por uma determinada chave.

4.3.2.2 Texto claro conhecido

Neste caso, o oponente tem conhecimento de vários pares de texto cifrado e claro. Este ataque é particularmente facilitado pela natureza repetitiva e bem conhecida dos cabeçalhos e demais estruturas dos protocolos de comunicação.

4.3.2.3 Texto claro escolhido

Este tipo de ataque é o mais poderoso em se tratando de criptoanálise do algoritmo de criptografia em si. O oponente, além de ter acesso a pares de texto cifrado e claro, é capaz de escolher blocos específicos de texto para serem cifrados, de maneira que estes possam induzir a informações a respeito da chave de ciframento.

A **criptoanálise diferencial** [19], introduzida em 1990 por Eli Biham e Adi Shamir, usa o princípio de escolher textos claros com diferenças particulares e analisar os textos cifrados produzidos, para derivar alguns bits da chave de algoritmos baseados no DES.

Criptoanálise linear [56] é outra técnica que utiliza pares de texto cifrado/claro para também derivar probabilisticamente partes da chave de ciframento, de algoritmos também baseados no DES.

Vale a pena ressaltar que, ataques de texto claro escolhido são mais fáceis de se implementar do que se poderia imaginar em uma primeira instância. Somente para citar um exemplo, este ataque pode ser facilmente aplicado por usuários de sistemas que empregam criptografia ponto a ponto, como no caso de redes de perímetro virtual (VPN).

4.3.2.4 Ataque do nó intermediário (*Man-in-the-Middle Attack*)

O controle de algum nó intermediário no percurso entre duas entidades, por parte de um oponente, é estrategicamente muito interessante. Se este for capaz apenas de grampear o meio de comunicação, isto é, for um oponente **passivo**, todos os blocos cifrados por um determinado algoritmo vão estar disponíveis para o processo de criptoanálise. Isto aumenta significativamente a probabilidade da criptoanálise ser bem sucedida.

Por outro lado, se o oponente tiver a habilidade de interceptar e retransmitir mensagens no meio de comunicação, ou seja, for um oponente **ativo**, ataques ainda mais sutis entram em questão. Como será abordado na seção que trata dos protocolos de gerenciamento de chaves, o nó intermediário pode, neste caso, personificar cada uma das extremidades (entidade) à outra.

Outro ataque possível é combinar duas mensagens cifradas com a mesma chave, para produzir uma terceira de significado distinto. Esta técnica é também conhecida como ataque de **recorte e colagem**. Finalmente, ataques baseados na **repetição de mensagens** são também mais facilmente implementados por oponentes que disponham do posicionamento estratégico descrito acima.

4.4 Funções de *hashing*

Frequentemente é interessante mapear um espaço potencialmente grande em outro de dimensão menor. Há bastante tempo esta idéia tem sido usada em estruturas de dados (*hash tables*) através de funções de *hashing*. Funções de *Hashing* apresentam a propriedade de que, independentemente do tamanho dos dados de entrada, esta retorna um valor de tamanho fixo. No entanto, para terem utilidade em sistemas de criptografia, estas funções devem apresentar uma segunda característica. Isto é, devem ser não inversíveis e deve ser impraticável de se encontrar duas mensagens que resultem em um mesmo valor de saída.

As funções que apresentam as duas características acima são normalmente denominadas de funções de *hashing* seguras, ou simplesmente funções de *hashing* para o contexto de estudo atual. Na literatura, os seguintes termos são também encontrados para descrever tais funções: função de compressão, função de contração, *message digest function*, *cryptografic checksum* e *message integrity check*.

Existe um número razoável de funções de *hashing* disponíveis ao público em geral. Dentre aquelas consideradas "seguras" destacam-se: MD2 [48], MD4 [74], MD5 [75] e SHA (*Secure Hash Algorithm*) [62].

A sigla "MD" refere-se a *Message Digest,* que designa as funções projetadas por Ronald Rivest, todas com valor de saída de 128 bits. A MD2, ainda que segura, apresenta problemas de desempenho em relação ao desejado para uma função do gênero. A versão MD5 é um aperfeiçoamento da função MD4 com base nos ataques sofridos por esta [12]. Recentemente, no entanto, a função MD5 foi submetida a criptoanálise semelhante a da MD4 [31]. Conseqüentemente, seu uso em sistemas a longo prazo é desaconselhado [89].

O SHA é uma função de *hashing,* projetada pela NSA[1] (*National Security Agency*) para ser usada no padrão de assinatura digital DSS (*Digital Signature Standard*). Além de produzir uma saída de 160 bits, o que aumenta a dificuldade de se implementar *birthday attacks* e ataques de força bruta, esta aparenta ser a opção mais segura disponível atualmente. A referência [76] apresenta uma comparação mais detalhada das funções descritas acima e também de outras não tão conhecidas.

1. Serviço secreto de segurança e criptografia dos EUA.

Funções de *hashing* podem também ser geradas através de cifradores de bloco como o DES. As senhas cifradas encontradas no arquivo de senhas da maioria dos sistemas UNIX (`/etc/passwd` ou `/etc/shadow`) são produzidas desta forma.

4.4.1 MAC (*Message Authentication Code*)

Grande parte das vulnerabilidades, encontradas nos modos de operação dos algoritmos de chave secreta, eram devidas à falta de um mecanismo que garantisse a integridade das mensagens enviadas. Funções de *hashing,* combinadas a uma chave secreta, podem ser usadas para produzir MACs, e conseqüentemente verificar a integridade destas mensagens.

O resultado de uma função de *hash* aplicada sobre uma mensagem concatenada a uma chave secreta, resulta em um valor que só pode ser computado pelas partes que conheçam a chave secreta usada.

4.4.2 *Birthday Attack*

Este ataque é baseado em um problema probabilístico clássico, conhecido como paradoxo da data de aniversário. A questão é: quantas pessoas são necessárias em uma sala para que a probabilidade de pelo menos uma ter uma determinada data de nascimento (a do leitor, por exemplo) seja maior que 50 por cento. A resposta é 183 pessoas. Por outro lado, se a pergunta for: quantas pessoas em média são necessárias em uma sala para que pelo menos duas possuam a mesma data de nascimento, a resposta é bem menor: 23 pessoas.

Analogamente, dada a saída de uma função de hashing de m-bits de comprimento, são necessárias em média 2^{m-1} mensagens aleatórias para produzir o mesmo resultado. Encontrar, no entanto, duas mensagens que produzam a mesma saída, requer somente $2^{m/2}$ mensagens aleatórias em média. Para evitar que este tipo de ataque seja factível em função do poder computacional atual, as funções de hashing devem ser projetadas para produzir valores de saída com tamanho superior a 128 bits.

4.5 Sistemas assimétricos ou de chave pública

Nos sistemas assimétricos a chave usada para cifrar uma mensagem é diferente da chave usada para decifrá-la ($K \neq K^{-1}$). Assim, uma chave pode ser publicada e outra mantida em segredo, for-

mando um par denominado **chave pública e chave privada**[1]. Para que este sistema seja seguro, o par chave pública e privada deve estar unicamente associado e ser impraticável derivar uma chave a partir da outra.

4.5.1 Notação

Seja $(\mathbf{K_p})_a$ uma chave pública e $(\mathbf{K_s})_a$ a chave privada associada de uma entidade A. Então escrevemos o ciframento e deciframento da seguinte forma:

$$C = (K_p)_a [P]$$

$$P = (K_s)_a [C]$$

Na prática, os algoritmos assimétricos são baseados em funções unidirecionais com escape (*trap-door one way functions*). Tais funções são implementadas através de problemas matemáticos computacionalmente intratáveis.

A primeira idéia neste sentido surgiu através do algoritmo de troca de chaves de Diffie-Hellman [35], proposto por Whitfield Diffie e Martin Hellman em 1976. Este algoritmo baseia-se na intratabilidade do problema do logaritmo discreto. Já o RSA [77], o mais conhecidos dos sistemas de criptografia assimétricos, tem por base a intratabilidade do problema de fatoração de números inteiros grandes.

Apesar de outros modelos de problemas terem sido propostos, tal como o problema da mochila [59], os sistemas mais conhecidos e importantes são baseados nos problemas matemáticos citados acima. É interessante observar a diferença na implementação entre os sistemas simétricos e os de chave pública. Os primeiros são baseadas em iterações de funções "elementares" de substituição e permutação, enquanto os sistemas assimétricos são baseados em problemas matemáticos de intratabilidade reconhecida.

De fato, os algoritmos assimétricos são implementados sobre grupos algébricos onde o problema do logaritmo discreto ou fatoração sejam intratáveis. A seção seguinte descreve o conceito de corpos e grupos algébricos finitos, bem como aqueles de interesse do ponto de vista de criptografia. Em seguida são descritos os algoritmos RSA, ElGamal e algoritmos baseados em curvas elípticas—ECC (*Elliptic Curve Cryptosystems*).

---

1. O termo "chave privada" foi empregado para diferenciar de "chave secreta" usado para designar as chaves de ciframento/deciframento dos algoritmos simétricos.

4.5.2 Conceitos algébricos básicos

O objetivo desta seção é oferecer os conceitos básicos e intuitivos para o entendimento de implementações reais de algoritmos assimétricos. Foge do escopo deste documento definir, ou tratar de maneira formal e detalhada, os conceitos matemáticos aqui descritos.

4.5.2.1 Grupos algébricos

Um grupo é constituído de um conjunto e uma operação entre os elementos desse conjunto com certas propriedades básicas: fechamento (todas as operações entre os elementos do grupo resultam em um elemento do mesmo), existência de inversos, existência de um elemento neutro e associatividade.

O grupo (Z_n, +)

O grupo (Z_n, +), ou simplesmente Z_n, é composto pelos inteiros de 0 a n-1. A operação básica do grupo é a adição módulo n. A adição modular é definida pelo resto da divisão por n do resultado da soma de dois elementos do grupo.

O elemento neutro do grupo Z_n é o zero (0). Cada elemento de um grupo aritmético possui um inverso tal que $x + x^{-1}$ (inverso de x) é igual ao elemento neutro do grupo.

No grupo Z_9, por exemplo, temos:

$7 + 6 = 4$, pois $(7 + 6) \bmod 9 = 4$

e o inverso de 4 é 5, pois $(4 + 5) \bmod 9 = 0$

O grupo (Z_p^*, *)

Este grupo é formado pelos inteiros entre 1 e p -1, onde p é, necessariamente, um número primo. A operação básica do grupo é a multiplicação modular. Como na adição modular, o resultado final é obtido pelo resto da divisão por p do produto da multiplicação dos outros dois elementos.

O elemento neutro do grupo Z_p^* é o elemento 1.

No grupo Z_{11}^*, por exemplo, temos:

$4 \times 7 = 6$, pois $(4 \times 7) \bmod 11 = 6$

e o inverso de 9 é 5, pois $(9 \times 5) \bmod 11 = 1$

Outras operações podem ser derivadas da multiplicação. Por exemplo, a exponenciação é definida como uma repetição da multiplicação. O inverso da exponenciação é o logaritmo discreto. Ou seja:

$7^3 = 2$, pois (7 x 7 x 7) mod 11 = 343 mod 11 = 2

e o logaritmo discreto de 2 na base 7 no Z_{11}^* é 3

A intratabilidade do logaritmo discreto em Z_p^* é a base para construção de diversos algoritmos assimétricos como, por exemplo, Diffie-Hellman, Elgamal e DSA. De fato, o grupo Z_p^* é um dos mais importantes do ponto de vista de sistemas de criptografia, sendo também a base algébrica para implementação do algoritmo RSA.

Outra propriedade importante no grupo Z_p^* é representada pelo conceito de elemento gerador do grupo g. Por definição, todos elementos do grupo podem ser expressos como uma potência de g.

No grupo Z_{11}^*, o elemento 2 é um gerador, pois:

$2^0 = 1$ $2^1 = 2$ $2^8 = 3$ $2^2 = 4$ $2^4 = 5$
$2^9 = 6$ $2^7 = 7$ $2^3 = 8$ $2^6 = 9$ $2^5 = 10$

4.5.2.2 Corpos algébricos

Corpos Algébricos são conjuntos munidos de duas operações aritméticas que obedecem a certas propriedades. Convencionalmente essas operações são chamadas de soma e multiplicação, embora outras operações sejam também admissíveis. Seguimos esta convenção no texto que se segue. Os elementos de um corpo compõem um grupo aditivo abeliano (grupo cuja operação básica é a comutativa), e os elementos não nulos formam um grupo multiplicativo abeliano. Desta forma, todos elementos apresentam um inverso multiplicativo e aditivo. Um corpo que apresenta um número finito de elementos é denominado corpo finito.

Os corpos finitos de interesse são o F_p e o F_2^m descritos a seguir.

Corpo F_p

O corpo F_p, onde p é um número primo, é composto pelos inteiros de 0 a p -1. As operações do corpo são a adição e a multiplicação modular definidas nos grupos Z_n e Z_p^* respectivamente.

Corpo F_2^m

O F_2^m é um corpo polinomial cujo interesse está em implementações de algoritmos em *hardware* e sistemas baseados em curvas elípticas (Seção 4.5.4.3, página 50). Existem diversas maneiras de representar a aritmética no F_2^m. A representação polinomial e a base normal ótima são as usadas na prática. A representação em base normal ótima é particularmente interessante para implementações em *hardware*. Nesta última representação, a operação de exponenciação pode ser calculada pelo deslocamento dos bits que representam o elemento a ser exponenciado.

Representação polinomial

Os elementos de F_2^m são definidos como polinômios de grau menor que m, cujos coeficientes pertençam ao corpo F_2 (0 e 1). Os elementos podem ser descritos em formato vetorial como $(a_{m-1},..., a_1, a_0)$, onde a_i é igual a 0 ou 1.

As operações básicas são a adição e a multiplicação modular definidas sobre um polinômio irredutível $f(x) = x^m + f_{m-1}x^{m-1} +... + f_2x^2 + f_1x + f_0$, ou seja, f(x) não pode ser fatorado em dois ou mais polinômios de grau menor que m sobre F_2.

A adição no corpo F_2 segue a seguinte fórmula $(a_{m-1},..., a_1, a_0) + (b_{m-1},..., b_1, b_0) = (c_{m-1},..., c_1, c_0)$, onde $c_i = a_i + b_i$ sobre F_2. Ou seja, OU-Exclusivo entre os coeficientes de mesmo grau do polinômio.

No grupo F_2^4, temos:

$$(0110) + (0101) = (0011)$$

A multiplicação é definida por: $(a_{m-1},..., a_1, a_0)(b_{m-1},..., b_1, b_0) = (r_{m-1},..., r_1, r_0)$, onde $r_{m-1}x^{m-1} +... + r_2x^2 + r_1x + f_0$ é o resto da divisão do produto dos polinômios $(a_{m-1},..., a_1, a_0)$ e $(b_{m-1},..., b_1, b_0)$ pelo polinômio irredutível f(x) sobre F_2.

No grupo F_2^4, com polinômio irredutível $f(x) = x^4 + x + 1$, temos:

$$(1101)(1001) = (1111)$$

pois:

$$= (x^3 + x^2 + 1)(x^3 + 1) \bmod f(x)$$

$$= (x^6 + x^5 + 2x^3 + x^2 + 1)(x^3 + 1) \bmod f(x)$$

$$= (x^6 + x^5 + x^3 + x^2 + 1)(x^3 + 1) \bmod f(x)$$

onde os coeficientes foram reduzidos módulo 2.

$$= (x^4 + x + 1)(x^2 + x) + (x^3 + x^2 + x + 1) \bmod x^4 + x + 1$$

$$= (x^3 + x^2 + x + 1) = (1111)$$

A exponenciação é também definida como uma repetição da multiplicação e o logaritmo discreto como o inverso da exponenciação.

4.5.3 RSA

O algoritmo RSA [77], cujo nome deriva das iniciais de seus autores Rivest, Shamir e Adleman, é certamente o algoritmo de chave pública mais conhecido e importante. Sua relevância em relação aos sistemas assimétricos equivale à do DES para os sistemas simétricos.

Como já mencionado, a segurança do algoritmo é baseada na intratabilidade do problema de fatoração de inteiros grandes (maiores que 100 dígitos). O RSA pode ser facilmente entendido pela compreensão de algumas propriedades básicas da aritmética modular e alguns resultados diretos da teoria dos números abordados a seguir.

4.5.3.1 Aritmética modular

Dizer que $a \equiv b \pmod n$, equivale a dizer que existe $k \in Z$, tal que $a = k \times n + b$. Diz-se que a é congruente a b, módulo n. Valem também as mesmas propriedades da aritmética convencional: comutatividade, associatividade e distributividade, isto é:

$$(a + b) \bmod n = (b + a) \bmod n,\ (a \times b) \bmod n = (b \times a) \bmod n$$

$$(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$$

$$(a \times b) \bmod n = ((a \bmod n) \times (b \bmod n)) \bmod n$$

$$(a \times (b + c) \bmod n = (((a \times b) \bmod n) + ((a \times c) \bmod n)) \bmod n$$

4.5.3.2 Teorema de Euler-Fermat

Generalização de Euler do **Teorema de Fermat**. Seja a função $\phi(n)$, n inteiro e positivo, definida como o número de elementos inteiros entre 1 e n-1, que são primos relativos de n. Isto é, tais que o máximo divisor comum (mdc) com n seja 1. Então, o teorema de Euler-Fermat diz que:

se $mdc(a,n) = 1$

Então:

$$a^{\phi(n)} \bmod n = 1$$

4.5.3.3 Parâmetros do RSA

O módulo de trabalho n, do RSA, é definido como o produto de dois números primos aleatórios grandes p e q, de mesmo tamanho aproximadamente. Assim,

$$n = p \times q$$

Neste caso, temos:

$$\phi(n) = (p - 1) \times (q - 1)$$

A chave de ciframento ***e***, é também escolhida aleatoriamente entre 2 e $\phi(n)$, tal que mdc $(e,\phi(n)) = 1$; a chave de deciframento ***d*** é o único inteiro entre 0 e $\phi(n)$ tal que:

$$ed \equiv 1 \pmod{\phi(n)}$$

A chave ***d*** pode ser facilmente obtida a partir de ***e*** e ***n*** pelo algoritmo estendido de Euclides [84].

4.5.3.4 Ciframento e deciframento no RSA

Para cifrar uma mensagem ***m***, esta é dividida em blocos ***b*** correspondendo a números inteiros menores que ***n***. A fórmula de ciframento é:

$$c = b^e \bmod n$$

e o deciframento:

$$b = c^d \bmod n$$

Isto porque:

$$c^d \bmod n = (b^e \bmod n)^d \bmod n = b^{ed} \bmod n$$

Mas:

$$ed = k\phi(n) + 1$$

Então:

$$b^{ed} \bmod n = b^{k\phi(n)+1} \bmod n = b^{k\phi(n)} b \bmod n = ((b^{k\phi(n)} \bmod n) \times (b \bmod n)) \bmod n =$$

$$((b^{\phi(n)} \bmod n)^k \bmod n) \times (b \bmod n)) \bmod n$$

Pela generalização do teorema de Fermat:

$$\mathbf{b^{\phi(n)} \bmod n = 1}$$

sempre que:

$$mdc\ (b,n) = 1$$

Daí:

$$((1 \bmod n) \times (b \bmod n)) \bmod n = b \bmod n$$

Ou seja, obtém-se a mensagem original ***b***. Esta demonstração pode ser estendida para os casos em que mdc(b,n) ≠ 1. Para maiores detalhes veja [94]. A Tabela 4.1 resume o algoritmo descrito acima.

Chave Pública	Chave Privada	Ciframento	Deciframento
n = p × q, p e q secretos	**d** = e^{-1} mod ϕ(n)	**c** = b^e mod n	**d** = c^d mod n
e, mdc (e, ϕ(n)) = 1	ϕ(n) = (p-1) × (q-1)		

Tabela 4.1: Tabela-resumo do algoritmo RSA.

4.5.4 ElGamal

O algoritmo Elgamal [37], proposto por Taher ElGamal, tem sua segurança baseada na intratabilidade do problema do logaritmo discreto sobre certos grupos aritméticos. Este algoritmo será descrito a seguir sobre o grupo $\boldsymbol{Z_p^*}$, no entanto outros grupos de interesse, como os formados pelos pontos de uma curva elíptica, serão também apresentados.

4.5.4.1 Parâmetros do ElGamal

Seja ***p*** um número inteiro primo que define um corpo $\boldsymbol{Z_p^*}$, e ***g*** um elemento gerador de $\boldsymbol{Z_p^*}$. Seja ***a*** um elemento de $\boldsymbol{Z_p^*}$, tal que

$$z = g^a \bmod p$$

onde ***p***, ***g*** e ***z*** são públicos (chave pública) e apenas ***a*** é secreto (chave privada).

4.5.4.2 Ciframento e deciframento no ElGamal

Para cifrar uma mensagem $\boldsymbol{m}$, esta é também dividida em blocos $\boldsymbol{b}$ correspondendo a números inteiros menores que $\boldsymbol{p}$. Além disso, um aleatório $\mathbf{k} \in \mathbf{Z_{p-1}}$ é usado para cifrar cada bloco de mensagem, tal que o texto cifrado é igual a:

$$c(b,k) = (X, Y)$$

onde

$$X = g^k \bmod p \text{ e } Y = b\, z^k \bmod p$$

O texto cifrado é decifrado através da seguinte relação:

$$d(X,Y) = Y\,(X^a)^{-1} \bmod p$$

Isto porque

$$Y\,(X^a)^{-1} \bmod p = b\,(z^k)(X^a)^{-1} \bmod p = b\,(z^k)(g^{ka})^{-1} \bmod n =$$

$$b\,(g^{ka})(g^{ak})^{-1} \bmod n = b \bmod n$$

Chave Pública	Chave Privada	Ciframento	Deciframento
p (primo), g (gerador)	x, aleatório, $\in Z_p^*$	**c** = (X, Y), onde: $X = g^k \bmod p$, $Y = bz^k \bmod p$	$d = Y/X^x \bmod p$
$z = g^x \bmod p$		k, aleatório, $\in Z_p$-1	

Tabela 4.2: Tabela-resumo do algoritmo ElGamal.

4.5.4.3 Sistemas de criptografia baseados em curvas elípticas

ECC (*Elliptic Curve Cryptosystems*) baseiam-se nas propriedades algébricas de grupos elípticos definidos por uma curva elíptica sobre outros corpos algébricos. A equação básica de uma curva elíptica satisfaz a seguinte fórmula:

$$y^2 = x^3 + ax + b$$

onde $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{a}$ e $\boldsymbol{b}$ são elementos do corpo algébrico no qual a curva é definida, e $4a^3 + 27b^2 \neq 0$.

Uma curva elíptica sobre os números reais é constituída de pontos cuja operação algébrica de adição é definida geometricamente conforme ilustra a Figura 4.4, página 51[1].

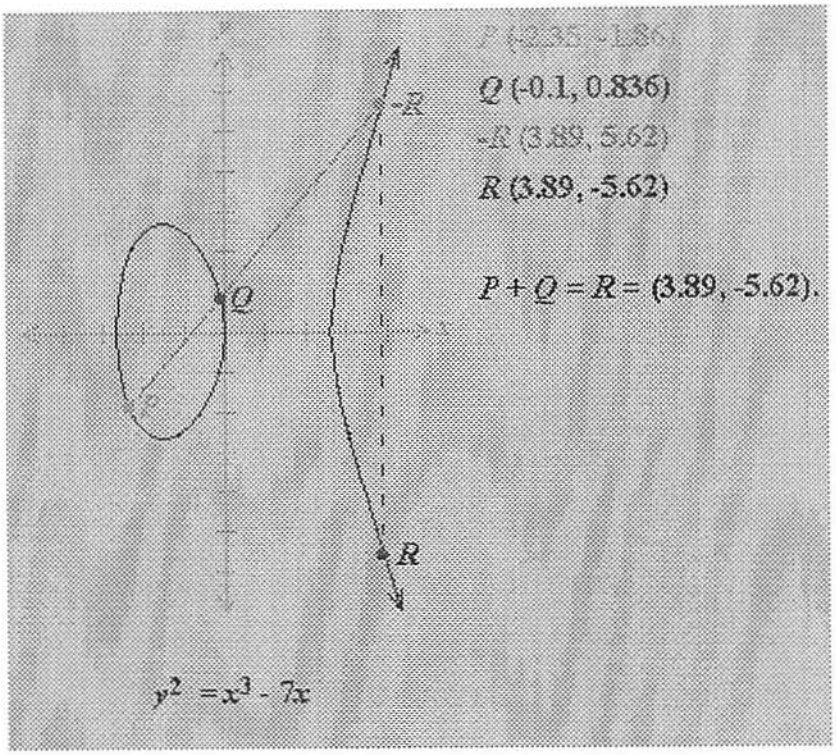

Figura 4.4: Operação aditiva entre dois pontos de uma curva elíptica sobre números reais.

Novamente, os grupos elípticos de interesse para a área de criptografia são formados por curvas elípticas sobre os corpos finitos $\mathbf{F_p}$ e $\mathbf{F_2m}$, onde y, x, a e b são respectivamente elementos de $\mathbf{F_p}$ e $\mathbf{F_2m}$. Os detalhes da aritmética destes grupos elípticos podem ser encontrados em literaturas mais especializadas no assunto [57], e são propositalmente omitidos neste documento.

4.5.4.4 O problema do logaritmo discreto em grupos elípticos

A segurança de ECC baseia-se na intratabilidade do problema do logaritmo discreto ou ECDLP (*Elliptic Curve Discrete Logarithm Problem*). A operação de Multiplicação Escalar nP, onde P é um ponto da curva e n um inteiro, é definida por n operações de soma algébrica do ponto P.

1. Exemplo extraído de [26].

O ECDP consiste em, conhecidos $\boldsymbol{P}$ e $\boldsymbol{nP}$, calcular $\boldsymbol{n}$. Na prática, $\boldsymbol{n}$ deve ser grande o suficiente para que o ECDLP seja de fato intratável.

Grupo	Z^*_p	ECC sobre F_p
Elementos do Grupo	Inteiros {1,2,3,.....,p-1}	Pontos (x, y) da Curva $\boldsymbol{E}$ onde x e y $\in F_p$ (ou F_2m)
Operação Aritmética	Multiplicação modulo p	Adição geométrica dos pontos da curva
Logaritmo Discreto	Dado g elemento gerador de Z^*_p, e $h = g^x \bmod p$, encontre x	Dado P um ponto de E, e Q = xP e x um número inteiro, encontre x

Tabela 4.3: Relação entre os Grupos Z*p e ECC sobre Fp.

4.5.4.5 RSA x ECC

Como o algoritmo RSA é de fato um padrão na indústria de informática, a questão que se aplica ao uso de sistemas baseados em curva elípticas não é somente se os ECC são de fato seguros, mas se estes são mais eficientes que os baseados no RSA.

Baseando-se na conjetura de que o logaritmo discreto sobre ECC é um problema mais difícil de ser tratado computacionalmente do que o logaritmo discreto no grupo $\mathbf{Z^*_p}$, ou que a fatoração de inteiros [47], pode-se obter o mesmo nível de segurança através de um grupo elíptico significativamente menor que um grupo $\mathbf{Z^*_p}$ correspondente. Por exemplo, um ECC definido sobre $\mathbf{F_p}$ com um ponto **P** cuja ordem é um primo de 160 bits oferece aproximadamente a mesma segurança que um sistema RSA cujo módulo $\boldsymbol{p}$ seja de 1024 bits [47].

Uma vantagem em trabalhar com grupos aritméticos menores está na possibilidade de implementar sistemas de criptografia em ambientes com recursos de *hardware* limitado como dispositivos de *smart cards*. Por outro lado, ECC constituem uma área de pesquisa mais recente que a fatoração de inteiros além de serem baseados em uma álgebra ainda mais complicada e menos intuitiva. De fato, a larga utilização de ECC ainda terá que percorrer um longo percurso de pesquisas e fóruns de padronização.

4.5.5 Sistemas simétricos X sistemas assimétricos

Nos sistemas simétricos, uma chave secreta deve ser estabelecida para cada par de entidades que queiram comunicar-se entre si. Tais chaves devem ser transmitidas de maneira segura, o que pode ser impossível, caso nenhuma comunicação segura tenha sido feita a priori.

Os sistemas assimétricos podem minimizar o problema acima, pois cada entidade interessada em receber mensagens secretas deve apenas publicar sua chave pública, em um diretório de comum acesso. Assim, para enviar uma mensagem sigilosa a esta entidade, basta obter sua chave pública via um canal de comunicação qualquer (inseguro). Somente a entidade que possuir a chave secreta correspondente será capaz de decifrar a mensagem em questão.

Por outro lado, os sistemas assimétricos não são usados diretamente para cifrar mensagens pelos seguintes motivos:

- São bastante lentos se comparados aos sistemas simétricos (cerca de 1.000 vezes mais lentos). Como os requerimentos de banda crescem proporcionalmente à velocidade das CPUs é improvável que estes sistemas possam ser usados para cifrar todo o fluxo de dados entre aplicações.
- Os sistemas assimétricos são mais vulneráveis a ataques de texto conhecido e escolhido. O acesso a textos cifrados conhecidos é extremamente facilitado pelo fato da chave de ciframento ser pública. Este problema se agrava quando o número de mensagens possíveis é reduzido. Neste caso, basta cifrar todas as mensagens possíveis com a chave pública disponível para identificar o conteúdo da mensagem enviada.

Os sistemas simétricos e assimétricos são na verdade complementares. Os de chave pública são normalmente empregados no processo de autenticação e ciframento de chaves de sessão, que por usa vez são usadas por algoritmos simétricos para cifrar o fluxo de dados entre aplicações. Sistemas de criptografia que se utilizam destas duas abordagens são frequentemente denominados **sistemas híbridos**.

4.5.6 Outros algoritmos assimétricos

O Apêndice I apresenta uma tabela-resumo dos principais algoritmos assimétricos, segundo julgamento dos autores deste trabalho.

4.6 Assinatura digital

Algoritmos de chave pública como RSA, ElGamal e DSS [63] (*Data Signature Standard*) podem ser usados para calcular a assinatura digital de uma determinada mensagem.

Assinar eletronicamente uma mensagem equivale, analogamente, a registrar em cartório o conteúdo de um dado documento (um contrato, por exemplo). A vantagem do sistema de assinatura digital, implementado por sistemas assimétricos, é que nestes não é necessário a figura de terceiros, como no caso dos cartórios. Pode-se ainda preservar a privacidade do conteúdo dos documentos assinados.

Algumas vezes, o algoritmo de chave pública empregado é usado para ao mesmo tempo cifrar e assinar um documento. Isto é possível em algoritmos tipo RSA, onde tanto a chave pública como a privada podem ser usadas no processo de ciframento e deciframento. Portanto, para uma entidade **A** assinar uma mensagem **M** e enviá-la secretamente até a entidade **B**, **M** deve ser cifrada da seguinte forma:

$$C = (K_p)_b\,[(K_s)_a[M]]$$

Somente a entidade **B** poderá decifrar a mensagem **C** com sua chave secreta, e conseqüentemente verificar, através da chave pública de **A**, a autenticidade da mesma.

O processo de geração de uma assinatura digital **S** é normalmente implementado com o auxílio de uma função de *hashing*. Ao invés de cifrar todo o conteúdo de uma mensagem, apenas a saída da tal função aplicada sobre a mensagem original é usada no procedimento acima. Em particular, a mensagem **M** acima seria assinada com o auxílio de uma função **H** da seguinte forma:

$$S_a = (K_s)_a\,[H[M]]$$

A implementação de assinaturas com funções de *hashing* oferece duas vantagens imediatas sobre o procedimento anterior. Primeiro, aumenta-se significativamente a velocidade de geração e verificação de assinaturas, já que as funções de *hashing* são sensivelmente mais rápidas de serem computadas que os algoritmos assimétricos em uso atualmente. Segundo, a assinatura pode ser gerada em separado da mensagem propriamente dita. Mais interessante: a assinatura é independente do fato da mensagem original ser mantida em sigilo ou não.

Capítulo 5

Gerenciamento de chaves

Neste capítulo serão apresentados os problemas de gerenciamento e distribuição de chaves, uma decorrência do uso de técnicas de criptografia em protocolos da Internet. Serão apresentados os principais algoritmos e seus aspectos de segurança.

5.1 Introdução

Possivelmente o principal obstáculo à difusão do uso de técnicas de criptografia, em redes de abrangência global como a Internet, é a falta de um mecanismo de gerenciamento de chaves em larga escala prontamente disponível. Felizmente, devido ao grande esforço da comunidade científica em geral, vários mecanismos foram desenvolvidos no decorrer dos últimos anos com tal objetivo.

Uma técnica comumente utilizada, principalmente nos sistemas simétricos (de chave secreta), é utilizar chaves específicas para cada sessão de comunicação entre dois nós. Geralmente **chaves mestras** (*master keys*) são usadas para distribuir ou gerar tais **chaves de sessão** (*session keys*). A problemática principal está normalmente focalizada em como estabelecer de maneira segura chaves mestras entre duas entidades quaisquer na Internet.

5.2 Distribuição manual de chaves

A maneira mais elementar de efetuar o estabelecimento de chaves é fazer com que cada usuário combine previamente uma chave mestra com cada um dos outros usuários, usando possivelmente um canal extra de comunicação seguro. O número de chaves estabelecidas será de $\mathbf{n \times (n-}$

1), onde **n** representa o número de entidades existentes na rede. Este problema é também conhecido como **problema do n^2** (*n^2 problem*) [84]. A despeito da simplicidade atingida, este mecanismo é excessivamente caro, para não dizer inviável, mesmo para redes com número de elementos relativamente pequeno.

A aplicabilidade desta técnica se resume a ambientes específicos como, por exemplo, um par de *firewalls* implementando uma rede de perímetro virtual (VPN). De qualquer modo, como veremos adiante, a RFC1825 [8], que especifica a arquitetura IP segura, exige que todas as implementações compatíveis com a mesma possuam gerenciamento manual de chaves.

5.3 Diffie-Hellman *Key Agreement Protocol*

A definição e a segurança matemática deste algoritmo [35], proposto por Whitfield Diffie e Martin Hellman em 1976, está na dificuldade em calcular o logaritmo discreto em corpos finitos, em contraste com a facilidade existente em exponenciar neste mesmo corpo (ver Seção 4.5.2, página 44). Informalmente, o algoritmo é definido da seguinte forma:

> Seja **g** um elemento gerador em $\mathbf{F_p}$;
>
> $\mathbf{g^x \bmod p}$ o elemento público[1] de uma entidade A, e **x** seu elemento secreto;
>
> $\mathbf{g^y \bmod p}$ o elemento público de uma entidade B, e **y** seu elemento secreto;
>
> Sendo que **x** e **y** são escolhidos aleatoriamente.
>
> A entidade A pode então calcular uma chave secreta, compartilhada apenas com B, a partir do elemento público de B e de seu elemento secreto da seguinte forma:
>
> $(g^y)^x \bmod p$
>
> Por raciocínio análogo, B calcula:
>
> $(g^x)^y \bmod p$
>
> Assim, ambos A e B obtém a mesma chave:
>
> $\mathbf{(g^x)^y \bmod p = (g^y)^x \bmod p}$

1. O termo "elemento público" é usado, ao invés de chave pública, para diferenciar da terminologia empregada em sistemas de criptografia assimétricos propriamente ditos (RSA, ElGamal etc).

Infelizmente, o algoritmo descrito acima está sujeito ao **ataque do nó intermediário** (*man-in-the-middle attack*). Isto ocorre quando não existe nenhuma maneira segura de associar uma entidade a seu elemento público. Quando a entidade A enviar uma mensagem requisitando o elemento público de B, esta mensagem pode ser interceptada por algum nó intermediário C, que por sua vez pode enviar seu elemento público como se fosse o de B. Procedimento análogo é em seguida efetuado sobre a requisição de B do elemento público de A. A e B têm a ilusão que estão se comunicando entre si, mas na realidade estão se comunicando com o intruso intermediário C.

5.4 Kerberos

O Kerberos é um sistema de autenticação e distribuição de chaves, projetado para operar em redes TCP/IP, baseado em um mecanismo proposto por Needham-Schroeder [79]. Neste protocolo, a autenticação é obtida através de um autoridade confiável, no caso, o próprio Kerberos.

O Kerberos é constituído de três elementos: TGS (*Ticket Granting Service*), KDC (*Key Distribution Center*) e usuários (aplicações cliente). O KDC compartilha uma chave secreta distinta com cada nó do sistema. Deste modo, ele é responsável pela autenticação de todos os clientes que queiram acessar um determinado servidor (serviço). Uma vez autenticados, os clientes têm que requerer um bilhete especial (*ticket*) ao TGS para poder acessar os servidores. Mais ainda, tais bilhetes só são concedidos caso o cliente tenha direito de acesso ao servidor requerido.

A seqüência de mensagens abaixo (ver também Figura 5.1, página 57) ilustra melhor o funcionamento do protocolo (versão 5) e dos elementos citados acima:

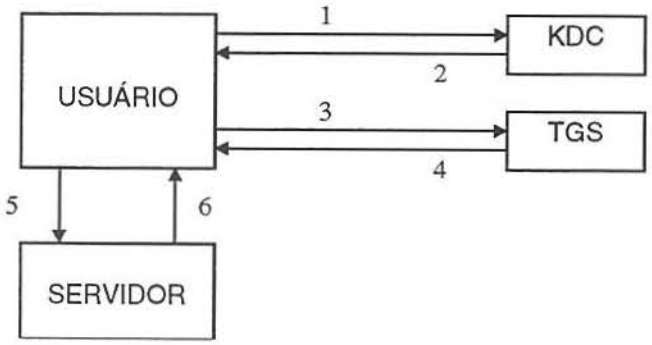

Figura 5.1: Fluxo de mensagens do protocolo Kerberos.

1. Requisição de bilhete de acesso ao TGS. O cliente envia sua identidade **(c)** e a do TGS ***(tgs)*** ao KDC

   C → KDC: c, tgs

2. Bilhete de acesso ao TGS. O KDC retorna ao cliente um bilhete de acesso ao TGS $(T_{c, tgs})$ cifrado com a chave do TGS e uma chave de sessão $(k_{c, tgs})$ cifrada com a chave secreta (k_c) do cliente:

   KDC → C: (k_c) $[k_{c, tgs}]$, (k_{tgs}) $[T_{c, tgs}]$

3. Requisição de bilhete de acesso ao servidor. O cliente envia para o TGS um autenticador (A_c) cifrado com a chave de sessão $(k_{c, tgs})$ e o anteriormente recebido bilhete de acesso ao TGS $(T_{c, tgs})$ cifrado com a chave secreta do TGS, juntamente com a identificação do servidor **(s)** a ser acessado:

   C → TGS: s, (k_{tgs}) $[T_{c, tgs}]$, $(k_{c, tgs})$ $[A_c]$

4. Aquisição do bilhete de acesso ao servidor. O TGS extrai a chave de sessão $(k_{c, tgs})$ do bilhete de acesso ao TGS $(T_{c, tgs})$ para decifrar e averiguar o autenticador A_c (maiores detalhes na Seção 5.4.1, página 59). A chave secreta do servidor (k_s) é usada para cifrar o bilhete de acesso ao servidor $(T_{c, s})$ e uma nova chave de sessão $(k_{c, s})$ para comunicação entre o cliente e servidor é enviada cifrada com a chave de sessão $(k_{c, tgs})$:

   TGS → C: (k_s) $[T_{c, s}]$, $(k_{c, tgs})$ $[k_{c, s}]$

5. Cliente apresenta bilhete de acesso ao servidor. O cliente apresenta ao servidor o bilhete emitido pelo TGS, que se encontra cifrado com a própria chave secreta do servidor, recebido na mensagem anterior. O cliente também envia ao servidor o autenticador (A_c) cifrado com a nova chave de sessão $(k_{c, s})$:

   C → S: (k_s) $[T_{c, s}]$, $(k_{c, s})$ $[A_c]$

6. Autenticação do servidor (opcional). Para serviços onde o cliente requer autenticação por parte do servidor, este deve retornar ao cliente o *timestamp* recebido no autenticador A_c acrescido de uma unidade:

   S → C: $(k_{c, s})$ **[ts + 1]**

5.4.1 Credenciais de autenticação (bilhete e autenticador)

Os bilhetes têm a função especial de autenticar o cliente junto ao servidor. Em outras palavras, garantir que o cliente que esteja usando um bilhete refere-se ao mesmo para o qual este bilhete fora emitido. A autenticação só pode ser confirmada se o bilhete estiver cifrado com a chave secreta do servidor, que apenas deveria ser de conhecimento deste e do TGS.

O nome do cliente **(C)**, o endereço IP **(IP_C)**, o nome do servidor **(S)**, a validade do bilhete **(lt)**, e a chave de sessão a ser utilizada para cifrar as próximas mensagens entre o cliente e o servidor são transmitidos ao servidor pelo cliente através do bilhete emitido pelo TGS e KDC para o cliente, conforme ilustrado abaixo:

$$T_{C,S} = S, (K_s)\ [C, IP_C, lt, K_{C,S}]$$

Uma vez obtido um bilhete, este pode ser usado para acessar o servidor (serviço) até que a validade do bilhete tenha se expirado (*lifetime*). Tanto o cliente como outras entidades na rede não são capazes de ler ou alterar o bilhete original emitido pelo TGS, pois como já mencionado, o mesmo encontra-se cifrado com a chave secreta do servidor.

Em contrapartida, um autenticador deve ser gerado toda vez que um cliente desejar acessar algum servidor (serviço). Sua principal função é evitar ataques de repetição de mensagens (enviando um *timestamp* para todas mensagens enviadas ao servidor) e autenticar o servidor perante o cliente (mensagem número 6). Diferentemente de um bilhete, o autenticador só pode ser usado uma única vez, já que pelo menos o *timestamp* ***(ts)*** deve ser incrementado a cada nova mensagem. O autenticador é cifrado com a chave de sessão entre o cliente e o servidor, como ilustrado a seguir:

$$A_{C,S} = (k_{C,S})\ [c, ts].$$

5.4.2 Análise de segurança

Infelizmente o Kerberos não resolve o problema original de distribuição automática de chaves mestras entre TGS, servidores e clientes. Novamente, algum mecanismo inicial deve ser responsável por definir as chaves secretas dos servidores e clientes no TGS. O uso de algoritmos assimétricos é uma solução proposta [13], mas que ainda não se encontra totalmente definida.

Apesar de ser bem mais seguro que os métodos de autenticação baseados no endereçamento IP e usuário UNIX, o Kerberos apresenta algumas vulnerabilidades próprias [18].

Oponentes que possuam acesso ao meio físico, podem colecionar os bilhetes em trânsito e posteriormente implementar **ataques de dicionário** (*password-guessing attacks*) nas chaves secretas dos clientes e servidores. Quanto maior o número de bilhetes coletados, maiores são as chances de sucesso para este tipo de ataque. Isto porque aumenta a probabilidade de se encontrar um bilhete cifrado com alguma senha "fraca", isto é, contida no dicionário do oponente.

Apesar de *timestamps* serem utilizados com intuito de evitar ataques de repetição de mensagens, autenticadores antigos podem ser reutilizados durante a vida útil de um bilhete, especialmente quando os servidores são incapazes de armazenar todos os bilhetes válidos que estão em uso e autenticadores já recebidos. De qualquer maneira, o ataque de repetição de mensagens só pode ser eliminado caso os relógios das entidades envolvidas (TGS, clientes, servidores) estejam sincronizados. No entanto, como a maioria dos protocolos de sincronização são inseguros [23], torna-se difícil garantir a segurança do protocolo neste sentido.

Contudo, o principal problema refere-se à segurança de sistemas pertencentes ao ambiente Kerberos, em particular em ambientes multi-usuários. Pouca segurança existe quando, por exemplo, algum usuário do sistema tiver acesso ao cache onde se encontram armazenadas as chaves de sessão de terceiros. Fragilidade maior ocorre quando o super usuário não for confiável. Nesta situação extrema, todo o sistema Kerberos pode ser substituído por um "cavalo de Tróia", suplantando toda a segurança do sistema em si.

5.5 Certificados públicos

A principal vulnerabilidade dos protocolos que utilizam algoritmos assimétricos no processo de distribuição de chaves decorre da ausência de uma maneira segura de associar uma entidade à sua chave pública. O **Ataque do nó intermediário**, tal como descrito para o algoritmo DH (Diffie-Hellman Key Agreement), pode nesta circunstância ser facilmente implementado.

Uma maneira de contornar este problema é através de **certificados públicos**. Estes certificados nada mais são que uma associação confiável de uma entidade com sua chave pública através de uma **assinatura digital** efetuada por um terceiro também confiável, denominado autoridade de certificação ou simplesmente CA (*Certificate Authority*).

Formalmente, todos os certificados devem apresentar pelo menos duas propriedades:

- oferecer uma ligação confiável entre um nome, uma entidade e sua chave pública

- prover integridade da chave pública desta entidade

É possível, então, dificultar as tentativas de substituir chaves públicas de uma entidade por outra. Naturalmente, alguma outra forma segura de obter a chave pública das CAs deve existir; provavelmente por algum outro método manual. De qualquer forma, tem-se uma redução significativa na escala do problema, especialmente se considerarmos, como veremos adiante, que tais certificados podem ser emitidos de maneira hierárquica.

5.5.1 Formatos dos certificados

A arquitetura TCP/IP ainda carece de uma infra-estrutura global para distribuição de certificados públicos. Basicamente os esforços têm se concentrado em três métodos: serviço de diretório X.500 [24], DNS Seguro (*Secure Domain Name System*) [36] e certificados PGP [11]. Na ausência de uma estrutura prontamente disponível, alguns protocolos específicos foram especialmente definidos para distribuição de certificados [3]. Desta forma, os certificados podem ser obtidos diretamente da entidade com a qual se queira comunicar, ou através de um servidor de dados central, sem a necessidade de nenhuma grande mudança estrutural.

Infelizmente cada um dos métodos propostos apresenta uma estrutura diferente. O certificado X.509 [25] do serviço de diretórios citado acima possui a seguinte estrutura de dados:

- ***Version*** - identifica versão do formato do certificado
- ***Serial Number*** - número seqüencial único para cada certificado emitido por uma CA
- ***Algorithm Id*** - algoritmo usado para assinar o certificado (p. ex. DSA ou RSA)
  - algoritmo - nome do algoritmo usado para assinar o certificado
  - parâmetros - parâmetros associados ao algoritmo em questão
- ***Issuer*** - nome (X.509 *Distinguished Name*) da CA
- ***Validity***
  - data de início de validade
  - data de término validade
- ***Subject*** - nome (X.509 *Distinguished Name*) da entidade cuja chave pública foi assinada
- ***Subject's Public Key***

  - algoritmo - nome do algoritmo associado à chave pública do proprietário do certificado
  - parâmetros - parâmetros associados à chave pública em questão
  - chave pública - atributos da chave pública propriamente dita
- ***Signature*** - assinatura gerada usando a chave pública da CA sobre os dados acima

5.5.2 Estrutura de certificação

Com o objetivo de reduzir ao máximo a necessidade de definir chaves manualmente, CAs podem ter também seus certificados assinados por uma autoridade de mais alto nível. Tem-se assim uma estrutura hierárquica em forma de uma árvore invertida. A raiz desta árvore representa a autoridade superior, responsável por delegar autoridade e assinar os certificados das CAs de nível imediatamente abaixo, e assim sucessivamente, até chegarmos aos usuários finais nas folhas da árvore.

Uma série de questões envolve uma estrutura deste gênero, principalmente em se tratando de uma rede de grande proporção. Dentre tais questões, podemos destacar algumas de especial interesse para a estrutura da Internet:

- Qual nível de confiança pode-se ter em uma CA?
- A quem será delegada Autoridade de Certificação (CA) em domínios globais (não locais)?
- Qual tipo de formalidade regerá a relação dos usuários com a sua CA, e a relação desta com a CA imediatamente superior?
- Haverá uma única raiz? A quem será delegada tamanha responsabilidade?

A RFC1422 [50], que especifica o gerenciamento de chaves da aplicação de correio eletrônico PEM (*Privacy Enhanced Mail*), é recomendada como leitura adicional ao leitor interessado neste tipo de problemática, bem como em certificados de uma maneira geral.

Podem existir situações onde uma estrutura hierárquica não é desejável. Notoriamente isto acontece quando os usuários não querem se submeter à estrutura de uma única raiz, estando portanto sob "custódia" de CAs distintas, ou mais ainda, quando não confiarem em nenhuma CA.

Para este tipo de situação deve haver alguma forma de estrutura linear distribuída capaz de permitir que qualquer entidade certifique a chave pública de outra. Outra aplicação de correio eletrônico segura, PGP [38], aborda este problema com o conceito de **apresentadores** (*intro-*

ducers). Apresentadores são simplesmente usuários que geraram certificados para outros usuários de sua confiança. Se um usuário confia em um determinado apresentador, este passa então a aceitar chaves públicas assinadas por este. Para que este sistema seja seguro, o apresentador deve de alguma forma ser capaz de conferir as chaves públicas das entidades que ele esteja endossando. Isto pode ser feito de forma manual (via telefone, ou pessoalmente), uma vez que é esperada do apresentador alguma proximidade com as entidades endossadas por este.

Esta cadeia de apresentações visa estabelecer, na verdade, uma teia de confiança (*web of trust*). Por exemplo, o usuário A, ao criar sua chave pública, envia uma cópia para seus amigos B e C, que passam então a ser seus apresentadores. A questão, no entanto, torna-se mais complexa em situações de apresentadores de apresentadores. Isto é, suponhamos que um usuário E foi apresentado a B por outro usuário D. Neste caso, E deveria confiar nas chaves assinadas (apresentadas) por B, em específico em A? É natural que na medida em que aumente a cadeia de apresentações, maiores sejam as chances de obter certificados não autênticos. O PGP, por exemplo, tem mecanismos de proteção que permitem estabelecer limites de validade a certificados, assinados por usuários que por si já foram indiretamente apresentados.

A maior vantagem deste último mecanismo é a ausência de Autoridades de Certificação (CA), e de todos os inconvenientes que estão agregados a elas, particularmente a necessidade de considerável infra-estrutura montada. Este talvez seja um dos motivos da maior popularização do PGP em relação ao PEM, justamente por este exigir uma estrutura hierárquica rígida.

5.5.3 Revogação de certificados

O gerenciamento de certificados seria bem mais simples caso a validade de uma chave pública não precisasse ser alterada, ou melhor, só perdesse a validade conforme o prazo indicado no seu próprio certificado (data de término da validade). Infelizmente pode ser necessário revogar um certificado antes deste prazo. Isto porque a chave pública de algum usuário foi de alguma forma comprometida, ou ainda porque a CA não deseja mais certificar um determinado usuário, talvez por este ter transgredido alguma norma imposta pela mesma.

Assim, toda CA deve manter uma lista de certificados revogados (CRL - *Certificate Revocation List*). Ao requerer um certificado a uma CA, o usuário deve verificar a CRL e possivelmente manter um cache local dos certificados revogados obtidos. Mais uma vez, a RFC1422 [50] apresenta-se como referência complementar ao gerenciamento de CRL.

O gerenciamento de CRL torna-se mais complicado para as estruturas distribuídas apresentadas acima, justamente por estas não possuírem nenhum gerenciamento centralizado. De qualquer forma, mesmo na presença de CA, existe a possibilidade de abuso deste sistema, especialmente em situações de atraso na propagação das CRL, ou quando uma CA estiver indisponível. Em geral, a revogação de certificados é a principal vulnerabilidade apresentada pelos mecanismos de gerenciamento de chaves baseados em certificados públicos.

Capítulo 6

Arquitetura IP segura

Neste capítulo serão apresentados os principais protocolos projetados com a preocupação de segurança, e adotados pela comunidade de desenvolvedores que mantém o TCP/IP. Análises de segurança também são feitas, na tentativa de qualificar a segurança efetivamente capaz de ser obtida com o uso de tais protocolos.

6.1 Proteção do enlace físico

A proteção mais direta e transparente que se pode ter em mente para qualquer sistema de comunicação é ter todo o nível de enlace cifrado (*link level encryption*). Assim, cada enlace é protegido individualmente por equipamentos específicos ou mesmo por um *device driver* apropriado. Ou seja, ter o meio de comunicação fisicamente protegido.

Apesar da transparência, a principal vulnerabilidade deste enfoque está nos nós intermediários de conexão dos enlaces, em específico nos roteadores. Para que haja comunicação segura entre dois pontos quaisquer, é necessário que todos os roteadores entre tais pontos sejam seguros, o que de maneira geral não ocorre.

Por outro lado, como até mesmo o endereço de origem e destino dos pacotes transmitidos pelos enlaces protegidos são cifrados, não é possível a um inimigo com acesso ao meio físico nem mesmo efetuar análise de tráfego. Este tipo de ataque baseia-se na inteligência de inferir fatos com base apenas na observação da origem e destino dos pacotes.

Logo, este método aplica-se principalmente na proteção de enlaces específicos e preferencialmente em conjunto com outro nível de ciframento de mais alto nível (aplicação) para fortalecer

o sistema de criptografia em uso e evitar análise de tráfego. A avaliação detalhada deste tipo de técnica e tecnologias empregadas fogem ao escopo deste trabalho.

6.2 Opções de segurança do protocolo IPv4 (IPSO)

As opções de segurança da versão atual do protocolo IP (IPv4) foram a única iniciativa original na época de definição do protocolo com relação à segurança do mesmo. Campos do cabeçalho IP foram reservados para (*IP security labels*) etiquetar (*labeled*) o nível de sensibilidade (*top-secret, secret, confidential, unclassified*) da informação contida no pacote IP [49].

Em linhas gerais um processo não pode escrever em um meio (ou compartimento) com um nível de segurança (sensibilidade) inferior, nem ler de um meio que contenha informação mais altamente classificada. Este tipo de consideração pode ser levada em conta por um roteador para impedir, por exemplo, que informações trafeguem por meios de comunicações não-condizentes com sua sensibilidade.

No entanto, o IPSO não especifica nenhum mecanismo que implemente os princípios de segurança definidos anteriormente. Trata-se mais de uma filosofia de compartimentalização dos pacotes em níveis de segurança distintos (*Multi Level Security* - MLS [33] [34]) do que um mecanismo de proteção em si.

As opções de segurança do protocolo IP definem uma política de segurança em estilo militar, e são principalmente usadas por instituições militares nos EUA e por algumas poucas implementações de sistemas operacionais (UNIX VMLS). A estrutura extremamente centralizada e altamente restritiva tornam o uso do IPSO pouco apropriado para o mundo comercial, onde as regras de segurança são menos severas que nos meios militares, e o controle normalmente descentralizado.

Ainda assim, o conceito de níveis de segurança pode ser construído na arquitetura IPSec, atribuindo um nível de segurança implícito à associação de segurança em questão. Isto deverá desestimular ainda mais o uso das opções de segurança no protocolo IPv4 (IP versão 4).

6.3 Arquitetura IP segura - IPsec

6.3.1 Introdução

A definição de uma arquitetura segura na camada de rede traz consigo certas vantagens, principalmente no que se refere ao legado de todas as aplicações já desenvolvidas. O principal objetivo é agregar conceitos de segurança (autenticação e privacidade por exemplo), de maneira transparente, a todas as aplicações do pacote TCP/IP.

Acrescenta-se também o fato desta arquitetura adaptar-se bem, como será abordado em maiores detalhes adiante, às topologias que fazem uso de *firewalls* e o seu papel na definição de redes de perímetro virtual (*Virtual Perimeter Network*—VPN).

Em contra partida à transparência e generalidade alcançados, o IPsec pode ser alvo de diversos ataques [95]. Apesar de representar um grande avanço em relação à estrutura convencional, as vulnerabilidades encontradas sugerem cautela no uso genérico e indiscriminado desta arquitetura.

6.3.2 Descrição geral

Uma das principais características desta arquitetura é o elevado nível de independência e modularidade entre os protocolos. Os conceitos de segurança (autenticidade, integridade, privacidade e não-repúdio), os algoritmos de criptografia usados nos protocolos (RSA, DES etc) e também os mecanismos de distribuição de chaves são independentes entre si.

Esta modularidade oferece grande flexibilidade na implementação de várias políticas de segurança, além de permitir o aperfeiçoamento e substituição de novos protocolos e algoritmos, sem no entanto, comprometer a arquitetura definida. Com o objetivo de garantir interoperabilidade, um conjunto de algoritmos padrões e transformações foram definidos. Dentre estas especificações destacam-se *keyed* MD5 [61] e DES CBC [53].

As restrições governamentais ao uso de criptografia estão normalmente relacionadas ao uso de técnicas de criptografia para obtenção de privacidade. As propriedades de autenticação, integridade e não-repúdio em comunicações não são normalmente proibidas na maioria dos países.

Neste contexto têm-se a definição de dois mecanismos de segurança. O primeiro, o *Authentication Header* (AH) [9] que oferece integridade e autenticação sem privacidade. O segundo, o

Encapsulating Security Payload (ESP) [10] oferece primordialmente privacidade. Naturalmente, esses mecanismos podem, se necessário, serem utilizados em conjunto.

Mais uma vez, devido à modularidade estrutural, uma série de protocolos de gerenciamento de chaves, tais como Photuris [54] e SKIP [2], puderam ser propostos e aperfeiçoados, independentemente do mecanismo de segurança utilizado. Como o gerenciamento eficaz de chaves constitui um dos maiores desafios na implementação de protocolos seguros, será dada nas seções seguintes especial atenção ao gerenciamento de chaves para a arquitetura IP.

6.3.3 Associações de segurança

Uma vez que os protocolos de gerenciamento de chaves e algoritmos de criptografia são independentes dos mecanismos de segurança propriamente ditos (AH e ESP), alguma forma de associação deve existir entre os mesmos.

Uma associação de segurança é definida pelo algoritmo, modo de operação (transformação), chave e demais propriedades de segurança (tempo de expiração, nível de sensibilidade dos dados etc) aplicadas sobre os pacotes processados pelos mecanismos de segurança em questão.

O parâmetro SPI (*Secure Parameter Index*) do cabeçalho dos protocolos AH e ESP, juntamente com o endereço do destinatário da mensagem, definem unicamente uma associação. Para evitar ambigüidade foi estabelecido ser de responsabilidade do destinatário a definição do valor do SPI a ser usado. Ou seja, para cada origem diferente o destinatário deve atribuir um SPI distinto. Formalmente uma associação reúne os seguintes parâmetros:

- algoritmo e modo do algoritmo de autenticação, quando AH estiver em uso (**necessário** em todas implementações AH)
- chaves utilizadas no algoritmo de autenticação (**necessário** em todas implementações AH)
- algoritmo de ciframento e modo de operação, quando o ESP estiver em uso. (**necessário** em todas implementações ESP)

- chaves utilizadas no algoritmo de ciframento (**necessário** em todas implementações ESP)
- vetor de inicialização (IV), quando requerido pelo modo de operação do algoritmo de ciframento (**necessário** em todas implementações ESP)
- chaves a serem usadas no algoritmo de autenticação de uma transformação ESP, caso esta transformação a requeira (**recomendado** para todas implementações ESP). O ESP pode também ser usado para autenticação em determinadas transformações
- tempo de expiração das chaves utilizadas (**recomendado** para todas implementações)
- tempo de expiração da associação (**recomendado** para todas implementações)
- endereço de origem da associação, pode ser um conjunto de endereços se mais de um sistema compartilha a mesma associação com um determinado destino (**recomendado** em todas implementações)
- nível de sensibilidade (*secret, classified etc.*) dos dados protegidos. Portanto, uma associação define implicitamente o nível de segurança dos dados, ao contrário do IPSO que a define de maneira explícita no cabeçalho IP (**necessário** para todos sistemas que possuam multi-níveis de segurança e **recomendado** para todos os outros sistemas)

As associações são normalmente unidirecionais. Sessões de comunicação entre duas máquinas terão normalmente um SPI para cada direção de tráfego.

6.3.4 *Firewalls* e *Virtual Perimeter Network*—VPN

Uma das vantagens de se definir uma arquitetura segura para o nível IP é a facilidade que se tem em integra-lá ao contexto de um *firewall*, em especial àqueles implementados em nível de rede [91]. Esta é certamente a principal aplicação a ser derivada da especificação do IPsec.

Filtros de pacotes, como são conhecidas estas implementações de *firewalls*, precisam ser capazes de atuar sobre o mesmo nível que os mecanismos descritos anteriormente, isto é, sobre os *drivers* da interface de rede. Desta forma é quase intuitivo imaginar que um mecanismo possa ser agregado ao outro em uma única implementação. Estar-se-ia associando à arquitetura segura a capacidade de efetuar filtragem de pacotes, o que é sem dúvida bastante conveniente.

Tal implementação pode ser efetuada basicamente de três maneiras, dependendo de como as associações com o *firewall* são abordadas. Isto equivale a dizer se existe ou não associações de segurança entre o *firewall* e a origem dos pacotes processados por este.

Quando o *firewall* não participa de nenhuma associação, este é incapaz de efetuar qualquer verificação relativa à criptografia sobre os pacotes processados. Os dados usados para efetuar o controle de acesso (origem, destino, protocolo de transporte, porta de origem e destino) sobre os pacotes não podem, portanto, ser verificados (autenticados via AH). A propagação (*proxing*) de mensagens SSL[1] é um bom exemplo da situação descrita acima.

Uma topologia interessante pode ser construída quando o *firewall* passa a efetuar as associações em nome das máquinas protegidas por este. Quando em uso com o AH, os dados utilizados no processo de controle de acesso e filtragem podem ser autenticados de maneira segura. Isto confere muito mais credibilidade ao processo de filtragem no *firewall*.

Organizações que possuem dois ou mais *firewalls* conectando redes fisicamente isoladas através da Internet podem fazer uso do ESP em cada ponto de interconexão para criar um túnel IP cifrado entre as máquinas da mesma organização. Em termos gerais, esta topologia implementa o conceito de rede de perímetro virtual (VPN).

A principal desvantagem desta topologia está justamente no papel centralizador do *firewall*. As máquinas internas ficam sujeitas à credibilidade do administrador do mesmo e à segurança da estrutura de comunicação interna (roteadores e meio físico), isto tudo sem levar em conta a própria segurança do *firewall*.

Para eliminar o problema da centralização de confiança no *firewall*, pode ser desejável que múltiplas averiguações de criptografia sejam efetuadas entre a origem e o destino de um pacote. Seria portanto necessário que associações fossem definidas em locais distintos, provavelmente uma no *firewall* e outra no destino final.

Obviamente mais de um *firewall* pode estar presente no caminho entre uma origem e um destino qualquer. Assim, o número de associações estabelecidas pode ser tão grande quanto possa tratar a escalabilidade do protocolo de gerenciamento de chaves utilizado.

---

1. Protocolo *Secure Socket Layer*, descrito em detalhes no Capítulo 7.

6.4 AH (IP *Authentication Header*)

6.4.1 Objetivo

O objetivo do AH é oferecer autenticação e integridade aos datagramas IP. Dependendo, no entanto, dos algoritmos de criptografia utilizados, pode ser possível obter também o conceito de não repúdio. O AH foi propositalmente definido para não oferecer privacidade, evitando assim possíveis restrições ao seu uso em países que proíbam tal prática.

6.4.2 Cálculo do MAC[1]

6.4.2.1 Algoritmo de autenticação

Algoritmos de criptografia fracos, tais como CRC-16, não devem de maneira alguma ser utilizados pelo AH. Ao invés disto, recomenda-se o uso de funções de *hashing* matematicamente seguras (MD5, SHA etc). Para manter a interoperabilidade é exigido que toda implementação do protocolo AH implemente a função MD5.

Independentemente do algoritmo utilizado, o MAC da mensagem é calculado sobre todo o datagrama IP. Isto inclui não apenas o cabeçalho IP, mas também todos os demais cabeçalhos de outros protocolos e os dados sendo transportados. Os campos e opções do cabeçalho IP, que podem ser modificados ao longo do caminho entre sua origem e destino, são preenchidos com zeros para efeito de cálculo pelo algoritmo.

6.4.2.2 Protocolo IPv4

Para o protocolo IPv4 apenas os campos *time to live* e *header checksum* precisam ser tratados de forma especial no cálculo do MAC da mensagem AH. Todos os demais campos, incluindo os referentes ao processo de desfragmentação IP [83] (*identification, flags* e *fragment offset*), são processados normalmente em função de seus valores originais. Desta forma, é mandatório que o processo de remontagem dos fragmentos IP aconteça antes do processamento local do AH.

Como os pacotes podem estar sujeitos à autenticação intermediária (em *firewalls*, por exemplo), é sugerido que implementações do IPv4 façam uso do protocolo *Path* MTU *Discovery*

1. *Message Authentication Code* definido na Seção 4.4.1, página 42.

quando o AH estiver em uso [58], evitando fragmentação ao longo do percurso a ser percorrido pelo pacote.

6.4.3 Formato do AH (*Authentication Header*)

O AH deve estar localizado logo após os cabeçalhos que são examinados por cada roteador intermediário entre uma origem e um destino qualquer, ou seja, após o próprio cabeçalho do protocolo IPv4, como mostram as figuras seguintes:

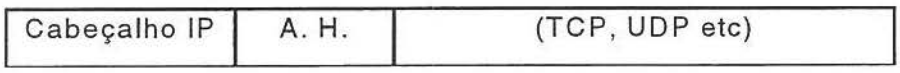

Figura 6.1: Localização do AH.

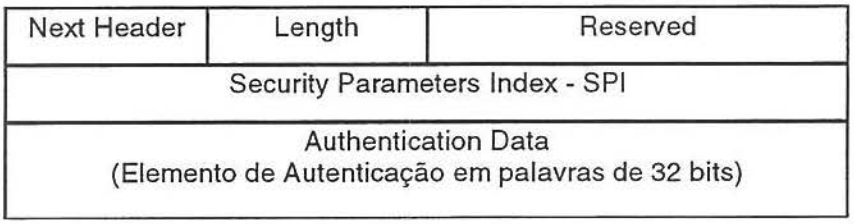

Figura 6.2: Descrição do AH.

- ***Next Header*** (8 bits) - identifica o protocolo que segue o AH. O valor para cada protocolo é definido em RFC pela IANA (*Internet Assigned Number Authority*) [82]
- ***Payload Length*** (8 bits) - comprimento em palavras de 32 bits do MAC
- ***Reserved*** (16 bits) - campos reservados para uso futuro; devem estar sempre preenchidos com zeros
- ***Security Parameters Index*** (SPI) - número de 32 bits que identifica a associação de segurança juntamente com o endereço IP de destino

- ***Authentication Data*** (n palavras de 32 bits) - **MAC**, que é resultado do algoritmo de autenticação aplicado sobre o cabeçalho e os dados do datagrama IP. Naturalmente, o conteúdo deste campo depende do algoritmo de autenticação usado pela associação em questão. Se o resultado do algoritmo não for múltiplo de 32 bits, alguma forma de preenchimento (*padding*) deve ser usada.

6.5 ESP (IP *Encapsulating Security Payload*)

6.5.1 Objetivo

O objetivo primordial do ESP é oferecer privacidade ao conteúdo dos dados encapsulados em um datagrama IP. Dependendo do algoritmo e o seu modo de operação (transformação), autenticação e integridade também podem ser oferecidos. No entanto, será mostrado adiante que tais transformações não são necessariamente seguras. Portanto, sempre que for requerido autenticação e/ou integridade, o AH deve ser utilizado em conjunto com o ESP.

6.5.2 Encapsulamento dos dados

Existem dois modos de encapsular os dados de um datagrama IP. No primeiro, conhecido como "modo de tunelamento", um datagrama IP completo é encapsulado dentro de um cabeçalho ESP.

Já no segundo modo de operação, apenas os dados do datagrama IP são encapsulados. Este modo é conhecido como "modo de transporte" porque, na maioria das vezes, os dados IP fazem parte de algum protocolo de transporte (TCP ou UDP). No entanto, nada impede que o ESP seja usado para encapsular protocolos de mais baixo nível, como por exemplo o protocolo ICMP (*Internet Control Message Protocol*).

Com o objetivo de manter a interoperabilidade na Internet, uma transformação específica para o encapsulamento dos dados, envolvendo o algoritmo de ciframento DES em modo CBC [53], foi formalmente definida. Esta transformação deve conseqüentemente fazer parte de qualquer implementação do mecanismo ESP.

6.5.2.1 Modo de tunelamento

A Figura 6.3 ilustra o ESP, sendo usado em um "túnel IP".

Este modo de funcionamento é particularmente interessante na criação de redes de perímetro virtual (VPN). O datagrama IP original, após ter sido cifrado e encapsulado via ESP, é colocado

Figura 6.3: ESP em Modo de Tunelamento.

novamente em outro datagrama IP. Os endereços de origem e destino do datagrama a ser enviado não precisam ser necessariamente idênticos aos do original. Isto possibilita ter alguma proteção, ainda que trivial, contra análise de tráfego. Os endereços de origem e destino usados neste datagrama podem, por exemplo, ser os de *firewalls* envolvidos na criação de uma VPN.

6.5.2.2 Modo de transporte

A Figura 6.4 ilustra o ESP sendo usado em "modo de transporte".

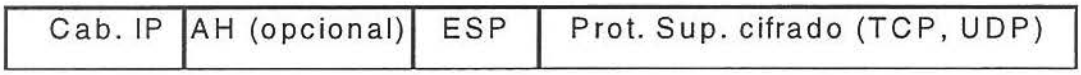

Figura 6.4: ESP em Modo de Transporte.

O segmento do protocolo de nível superior (TCP, UDP ou ICMP) extraído do datagrama IP é cifrado e, então, encapsulado após o cabeçalho ESP do datagrama IP original.

6.5.3 Autenticação no mecanismo ESP

Como já observado, em situações onde os conceitos de autenticação e integridade forem desejados, é extremamente recomendado o uso do AH em conjunto com o ESP para obter tais propriedades. Dependendo de quais dados se queira autenticar, se todo o datagrama IP ou apenas a parte cifrada no ESP, a localização do AH pode conseqüentemente variar.

Para autenticar todo o datagrama IP transportado, o AH deve ser posicionado antes do ESP. O MAC deve ser calculado sobre tanto os dados cifrados no ESP, como sobre os textos claros dos campos do próprio cabeçalho IP. Obviamente, o processo de decifragem dos dados protegidos via ESP só deve ser efetuado caso a autenticação tenha sucesso no destino final do datagrama.

Se apenas os dados protegidos pelo ESP, em modo de tunelamento, devem ser autenticados, o AH deve estar localizado após o cabeçalho IP do datagrama original a ser encapsulado (dentro do ESP). Naturalmente, o MAC é calculado sobre este mesmo datagrama antes deste ser cifrado.

Autenticação adicional pode ser derivada em pontos distintos, caso os dois enfoques descritos acima sejam usados em paralelo. Pode ser interessante autenticar todo o datagrama no *firewall* e os dados encapsulados no ESP apenas no seu destino final, após este ter sido decifrado. Neste caso, dois cabeçalhos AH seriam processados nos moldes descritos, respectivamente, nos parágrafos anteriores.

Esta dupla autenticação só seria possível caso não haja fragmentação IP, já que tanto o AH como o ESP só podem ser processados após a completa remontagem dos fragmentos IP.

6.5.4 Formato do ESP (*Encapsulating Security Payload*)

O ESP deve estar localizado logo após os demais cabeçalhos IP, incluindo o próprio AH, conforme ilustrado a seguir:

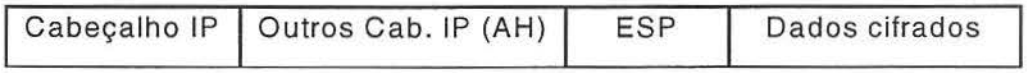

Figura 6.5: Localização do Cabeçalho ESP.

A figura seguinte, por sua vez, ilustra em maiores detalhes o formato do cabeçalho ESP:

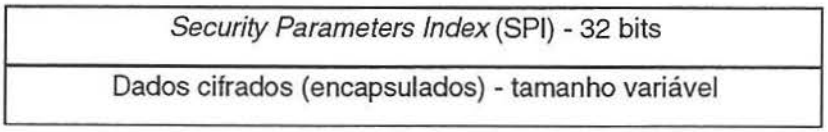

Figura 6.6: Cabeçalho ESP.

O campo SPI, cuja finalidade é a mesma que no mecanismo AH, identifica juntamente com o endereço de destino a associação responsável por determinar a transformação a ser aplicada sobre os dados contidos no campo seguinte.

Além dos campos mostrados acima, podem existir outros, específicos para uma determinada transformação. O SPI é o único campo obrigatório e independente da transformação em uso.

6.6 Análise de segurança

A definição da arquitetura IPsec representa, sem dúvida, um novo patamar de segurança para a estrutura do conjunto de protocolos TCP/IP. Entretanto, o IPsec está sujeito a uma série de outras vulnerabilidades que, porém, são bem mais sutis e sofisticadas.

6.6.1 Algoritmos de criptografia

Deve ficar claro que a segurança dos mecanismos até então apresentados depende diretamente i) da qualidade dos algoritmos de criptografia escolhidos; ii) do tamanho e qualidade das chaves adotadas; iii) da segurança dos protocolos de gerenciamento de chaves em uso; iv) e da correta implementação e depuração dos elementos mencionados acima.

6.6.2 Sistema operacional

A segurança do sistema operacional e programas de suporte em uso são de crucial importância para a segurança de qualquer sistema de criptografia. Isto porque, na maioria das vezes, estes são responsáveis, dentre outras coisas, por manter de maneira segura as chaves de sessão e privadas em memória.

A importância da segurança do sistema operacional aumenta em ambientes multi-usuários, onde sempre podem existir pessoas mal intencionadas, interessadas em quebrar a segurança do sistema. A existência de cavalos de Tróia e outras ameaças do gênero [101] constituem também uma preocupação constante neste tipo de ambiente.

6.6.3 Ataques baseados em recorte e colagem

Ataques de recorte e colagem sobre o fluxo de dados da arquitetura IP segura é conseqüência das seguintes vulnerabilidades:

- propriedades intrínsecas do modo de operação dos algoritmos de criptografia, em especial as do modo CBC
- ausência de mecanismos específicos para verificação de integridade dos dados transmitidos (AH ausente)

- gerenciamento de chaves definido por máquina, ou seja, todos os usuários de uma determinada máquina compartilhando a mesma chave para um mesmo destino; neste caso, ataques de texto escolhido (*chosen plain text attack*) podem ser facilmente efetuados por usuários locais

Quando factível, este tipo de ataque pode ser usado para decifar o conteúdo dos pacotes IPsec e até mesmo para sequestrar sessões de protocolos superiores. Na verdade, uma série de outras vulnerabilidades tornam-se também expostas [95].

Um ataque de recorte e colagem de mensagem pode ser facilmente implementado se o gerenciamento de chaves for definido por máquina, ou seja, se todos os usuários compartilham a mesma chave de sessão, e caso um mecanismo explícito de autenticação (MAC) não esteja sendo usado.

O intruso deve interceptar uma mensagem legítima e combiná-la com uma outra mensagem enviada por este anteriormente. A idéia é compor uma terceira mensagem a partir da original e a montada pelo intruso, preservando as informações confidenciais da primeira (por exemplo senha ou *cookie* de autenticação), e alterando blocos específicos que foram maliciosamente produzidos na mensagem enviada pelo intruso. Estes blocos podem ser partes da mensagem que contenham informação cuja alteração beneficie o intruso, como por exemplo o valor de um depósito bancário. Esta mesma técnica pode também ser usada para implementar seqüestros de sessão [95].

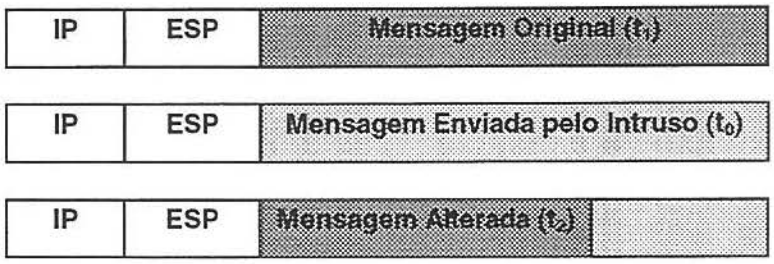

Figura 6.7: Ataque de Recorte e Colagem.

Note que ambas as mensagens foram cifradas com a mesma chave de sessão, e que é necessário ao intruso um conhecimento detalhado da estrutura da mensagem original enviada. Devido às propriedades auto-regenerativas do modo CBC, somente o primeiro bloco da parte da men-

78 ESTA FALTANDO

Por outro lado, tanto a criptoanálise diferencial [19] [22] como a maioria das vulnerabilidades aqui apresentadas são em grande parte devidas à facilidade que se tem para efetuar ataques de texto conhecido e escolhido. A melhor maneira de eliminar esta fraqueza é evitar o gerenciamento de chaves por máquina. Idealmente, uma nova chave de sessão deveria ser definida para cada conexão de usuário (processo) aberta.

6.7 Gerenciamento de chaves na arquitetura IP segura

Uma questão abordada com relação aos mecanismos AH e ESP, dizia respeito à ausência de especificação para o gerenciamento de chaves na definição da arquitetura IP segura. Os objetivos deste desacoplamento eram permitir a implementação de diversos tipos de mecanismos de distribuição de chaves distintos, como também a fácil substituição destes por outros mais atuais e supostamente melhores.

Uma possibilidade é ter o estabelecimento de associações de segurança fora do canal IP de dados, ou seja, fora da banda de dados (*out of band* - OOB). As informações referentes ao estabelecimento de chaves seriam transmitidas através de algum mecanismo de mais alto nível, sobre UDP ou TCP, em uma sessão à parte.

O estabelecimento de associações de segurança fora da banda de dados permite produzir mecanismos de distribuição de chaves tão sofisticados quanto se queira. Por outro lado, deve ser computado um atraso na comunicação (*delay*), toda vez que um novo par de chaves tiver que ser definido. O protocolo Photuris é um exemplo de mecanismo, ainda em fase de especificação por parte da IETF (*Internet Engineering Task Force*) [54], que segue os conceitos tratados acima.

Outros mecanismos, como o protocolo SKIP [2], utilizam a própria banda do canal de comunicação do protocolo IP para transportar informações necessárias ao estabelecimento de associações de segurança.

6.8 *Simple Key Management for Internet Protocols* (SKIP)

Como indica a própria denominação, trata-se de uma estrutura simples de gerenciamento de chaves. Por ter sido projetada para protocolos orientados a datagramas, como o IP, o gerenciamento

de chaves ocorre no nível de pacotes, na própria banda de dados do protocolo. Isto difere o SKIP da maioria dos mecanismos de gerenciamento de chaves que são normalmente orientados à sessão.

De fato, o SKIP não requer nenhuma comunicação prévia entre entidades, para o estabelecimento de um canal seguro de comunicação através dos mecanismos AH e/ou ESP. Isto proporciona uma redução no atraso de comunicação (*delay*) às custas de uma deterioração no uso da largura de banda, já que todo pacote transmitido deve conter, dentre outras coisas, uma cópia cifrada da chave de sessão e demais parâmetros da sessão em questão.

6.8.1 Descrição geral

O SKIP utiliza basicamente dois conceitos anteriormente abordados: DH *Key Agreement Protocol* e certificados assinados (ver Seção 5.5, página 60), usados para obter e autenticar os elementos públicos (chave pública) do algoritmo anterior.

O formato dos certificados, bem como sua infra-estrutura de distribuição, podem variar de acordo com o padrão escolhido: certificados X.509, certificados PGP, e DNS Seguro etc. Na ausência de uma infra estrutura de distribuição universal, tal como um serviço de diretórios X.500, foi definida uma estrutura própria através da especificação *Certificate Discovery Protocol* [3]. Este protocolo destina-se à obtenção de certificados assinados de um elemento qualquer na rede. Conforme determina a RFC1825 [8], são definidos também procedimentos manuais para o estabelecimento de chaves públicas.

Partindo do pressuposto de que cada entidade conheça o elemento público DH do destino ao qual esta queira se comunicar, uma chave mestra de longa duração $\mathbf{K_p}$ pode ser automaticamente calculada por este par de entidades. De acordo com o demonstrado na Seção 5.3, página 56:

$$K_{xy} = (g^x)^y \bmod p$$

onde $\mathbf{g^x \bmod p}$ e $\mathbf{g^y \bmod p}$ são os elementos públicos destes nós.

Finalmente, uma chave de sessão $\mathbf{K_s}$, gerada aleatoriamente, é usada para cifrar o datagrama IP, de acordo com o especificado pelos mecanismos ESP e/ou AH. A chave de sessão $\mathbf{K_s}$ é então cifrada com a chave mestra $\mathbf{K_{xy}}$. Apesar de cada pacote ter sua própria chave de sessão, não é

necessário que estas sejam distintas para cada pacote. As chaves podem ser mudadas de acordo com o definido na política de gerenciamento de chaves.

Ao receber pacotes associados ao protocolo SKIP, o primeiro passo a ser efetuado pelo receptor do pacote é obter e verificar o certificado DH do emissor do pacote. De posse do elemento público do originador da mensagem, a chave mestra $\mathbf{K_{xy}}$ é computada e usada para decifrar a chave de sessão $\mathbf{K_s}$. Finalmente a chave de sessão $\mathbf{K_s}$ é usada para decifrar o restante do pacote IP recebido.

A Figura 6.8, ilustra um pacote SKIP/AH/ESP:

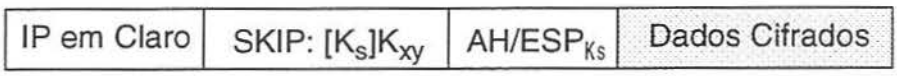

Figura 6.8: Pacote SKIP/AH/ESP.

6.8.2 Geração da chave mestra K_{xy}

Na verdade, a chave $\mathbf{K_{xy}}$ é derivada dos bits de mais baixa ordem da chave secreta definida pelo algoritmo DH. Isto porque o valor computado pelo algoritmo DH deve ser superior atualmente a 1024 bits para que o mesmo seja seguro, e as chaves dos sistemas de criptografia simétricos normalmente não ultrapassam esta cifra. Tipicamente temos o DES com chaves de 64 bits, DES triplo com três de 64 bits e IDEA com chaves de 128 bits [79].

Do ponto de vista de segurança do sistema, é importante que a chave mestra seja alterada com uma certa freqüência. Primeiro para dificultar a criptoanálise da chave mestra, diminuindo sua exposição. Segundo, porque elimina a possibilidade de uso de chaves de sessão comprometidas e ataques de repetição de mensagens.

No entanto, para alterar a chave $\mathbf{K_{xy}}$ é necessário revogar um certificado ou que o mesmo tenha sua validade vencida. O conceito de certificados, no entanto, pressupõe que estes possuam uma certa durabilidade. A maneira encontrada para atualizar $\mathbf{K_{xy}}$, sem atualizar certificados, foi através da seguinte relação:

$$K_{xyn} = h(K_{xy}\ n),$$

onde **n** é um contador e **h** uma função de *hashing* qualquer.

O contador **n** deve ser sempre incrementado. Este pode ser o número de unidades de tempo decorridas desde a última chave mestra calculada. A granularidade da unidade de tempo pode ser grande, variando até unidades de horas.

6.8.3 Formato do cabeçalho SKIP

A Figura 6.9, página 82, ilustra o cabeçalho SKIP, seguida pela descrição de cada um dos campos envolvidos:

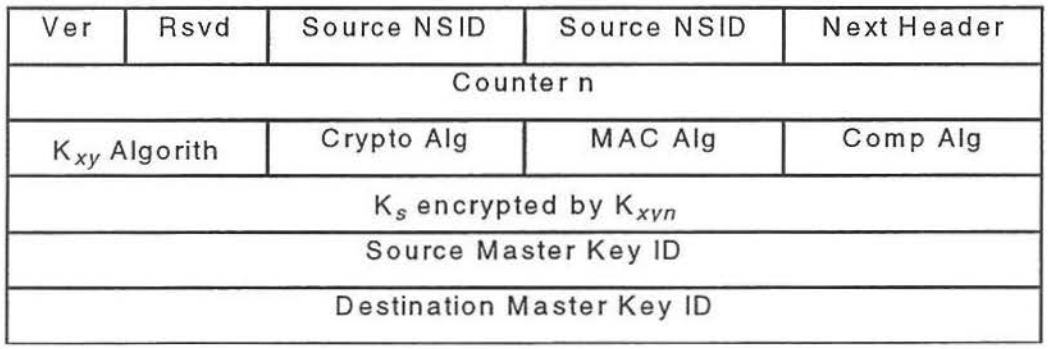

Figura 6.9: Cabeçalho SKIP.

- **Ver** (4 bits) - versão do protocolo SKIP
- **Rsvd** (4 bits) - campos reservados para uso futuro; devem estar sempre preenchidos com zeros
- ***Source* NSID** (8 bits) - especifica o espaço de endereçamento usado para identificar a entidade de origem (*Master Key-ID*)
- ***Destination* NSID** (8 bits) - especifica o espaço de endereçamento usado para identificar a entidade de destino (*Master Key-ID*)
- ***Next Header*** (8 bits) - identifica o protocolo que segue o SKIP; normalmente AH e/ou ESP
- ***Counter n*** (32 bits) - contador utilizado para gerar a chave mestra $\mathbf{K_{xyn}}$
- $\mathbf{K_{xy}}$ ***Algorithm*** - algoritmo usado para cifrar $\mathbf{K_s}$ através de $\mathbf{K_{xy}}$
- ***Crypto*** e **MAC Algorithm** - algoritmos usados pelo protocolo interior (AH e/ou ESP) para ciframento e autenticação

- $\mathbf{K_s}$ ***encrypted by*** $\mathbf{K_{xy}}$ (tipicamente 8-16 bytes) - chave de sessão cifrada pela chave mestra
- ***Source*** e ***Destination Master Key-ID*** - respectivamente, nome da entidade origem e entidade de destino, conforme endereçamento determinado pelos NSIDs

6.8.4 Associações de segurança

A especificação da arquitetura IP segura determina que o SPI (*Security Parameter Index*) de uma associação de segurança é uma atribuição do destinatário de uma mensagem. Como o protocolo SKIP não requer comunicação prévia para estabelecimento de um canal de dados seguro, o valor do SPI deve ser estabelecido ***a priori***.

Conforme especificado pela referência [2], foi estabelecido pela IANA (*Internet Assigned Number Authority*) que o valor "1" (um) é reservado para associações de segurança que façam uso do protocolo SKIP.

Note-se que, neste caso, o SPI e o endereço de destino não determinam unicamente uma associação de segurança conforme especificado na Seção 6.3.3, página 68. Esta ambigüidade é resolvida através do KeyID de origem e destino contidos no cabeçalho SKIP, que para o mecanismo SKIP determinam unicamente uma associação de segurança.

6.8.5 Análise de segurança

6.8.5.1 Ataques de dicionário e geração aleatória de chaves

Uma vez que as chaves de sessão $\mathbf{K_s}$ são geradas de maneira aleatória, é praticamente impossível aplicar ataques de dicionário sobre estas. Isto é válido mesmo se a chave mestra for definida manualmente de forma "fraca", ou seja, for uma palavra de um dicionário conhecido.

Portanto, uma das características de implementação mais importantes para o protocolo SKIP refere-se ao projeto de funções pseudo-aleatórias não previsíveis. Qualquer falha neste sentido pode comprometer toda a segurança do protocolo em si. A verdade é que a geração de seqüências aleatórias é um tópico relativamente mais complexo do que aparenta a uma primeira vista. A RFC1750 [28] é recomendada como leitura adicional a todos aqueles interessados em implementar protocolos que exijam aleatoriedade, tal como apresentado no parágrafo acima.

6.8.5.2 Ataque de texto conhecido ou escolhido

Nomenclatura original em inglês: ***Known and Chosen Plain Text Attack***. Independentemente do número de chaves de sessão comprometidas, este fato não representa nenhuma facilidade para determinar a chave mestra em uso. Além disso, sem conhecer a chave mestra, as chaves de sessão comprometidas não têm como ser reutilizadas em ataques futuros.

O fato de conhecer chaves de sessão não ajuda a determinar a chave mestra, simplesmente porque ter conhecimento ou escolher chaves de sessão a serem cifradas equivale a um ataque de texto conhecido/escolhido sobre uma chave $\mathbf{K_{xyn}}$ e não sobre a chave $\mathbf{K_{xy}}$.

6.8.5.3 Negação de serviço

O cálculo de chaves secretas compartilhadas pelo algoritmo DH é uma atividade computacionalmente intensa. Conseqüentemente há um grande potencial para ataques de negação de serviço requisitando-se, propositalmente, um número excessivo de cálculo de chaves mestras pelo algoritmo DH.

O SKIP ameniza este tipo de ataque, pré-calculando e armazenando as chaves mestras $\mathbf{K_{xy}}$ de todos os nós com que este tenha uma certa "afinidade". A relação de "afinidade" pode ser definida com base na freqüência de uso, ou por interferência manual. Além disso, o processo de comunicação tem preferência sobre os cálculos de chave mestra. Desta forma, é possível haver comunicação com uma máquina cuja chave mestra já esteja pré-computada, mesmo que um ataque de negação de serviço esteja em curso.

6.8.5.4 Sincronismo de relógio

A atualização da chave mestra através de um contador ***n*** requer um sincronismo de relógio na mesma granularidade que for desejado o incremento do mesmo. Na melhor das hipóteses, este mecanismo oferece grande oportunidade para implementação de ataques de negação de serviço, alterando-se o contador **n** do cabeçalho SKIP. Desta forma, é desejável que algum protocolo confiável de sincronismo de relógios esteja disponível juntamente com o protocolo SKIP.

6.9 IPv6

6.9.1 Introdução

A versão atual do protocolo IP (versão 4), definida no final dos anos setenta, já apresenta algumas limitações em função da realidade tecnológica da época de sua definição. Sua principal deficiência refere-se à limitação do espaço de endereçamento IP (endereços IP de origem e destino). Os 32 bits da versão corrente, destinados ao endereçamento IP, não suportam o tamanho atual e potencial de crescimento da Internet.

Outros fatores, como o suporte à aplicações de tempo real (*real-audio*, por exemplo) com demanda por baixas taxas de atraso (*low delay*), têm motivado a definição de uma nova geração do protocolo IP. Em particular, esta nova versão do protocolo IP deveria incluir também mecanismos nativos para autenticação e segurança.

Antes de abordar os mecanismos de segurança específicos do protocolo IPv6[1] (versão 6), as características básicas do IPv6 serão apresentadas. Conforme descrito em [29] as principais mudanças inseridas foram:

- espaço de endereçamento de 128 bits - o endereço no IPv6 é de 128 bits contra 32 bits da versão atual
- cabeçalho flexível - o IPv6 usa um formato de cabeçalho modular com possibilidade de adição de funcionalidade específica; por exemplo, o cabeçalho de Autenticação (AH)
- alocação de recursos - mecanismos de suporte e garantia de serviços de largura de banda e atraso (*delay*) foram inseridos
- extensão - talvez a principal característica do IPv6 seja sua estrutura modular que permite a extensão de funcionalidade e novos cabeçalhos associados

6.9.2 Formato do cabeçalho IPv6

Uma das principais mudanças do IPv6 em relação ao IPv4 está no formato de seu cabeçalho. O IPv6 possui um cabeçalho base de tamanho fixo, seguido de um ou mais cabeçalhos de extensão (*extension headers*). As Figuras 6.10 e 6.11 ilustram, respectivamente, a estrutura geral e a parte fixa do novo cabeçalho IP, seguidas da descrição dos seus campos:

1. As versões 1 a 3 nunca foram oficializadas e a versão 5 foi atribuída a outro protocolo denominado ST.

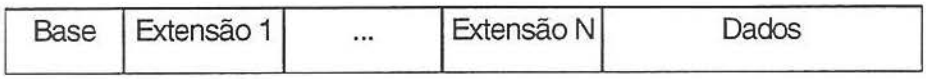

Figura 6.10: Cabeçalho IPv6 genérico.

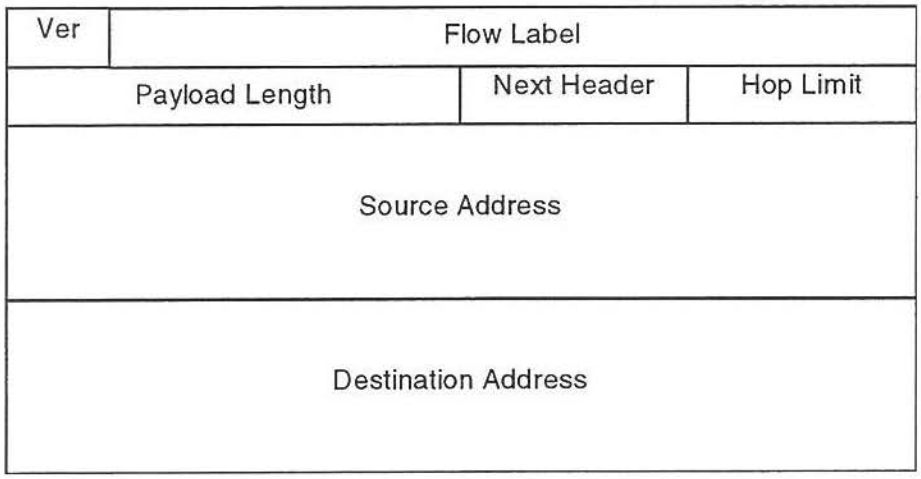

Figura 6.11: Cabeçalho base (fixo) de todo datagrama IPv6.

- **Ver** (4 bits) - versão do protocolo IP
- ***Flow Label*** (28 bits) - contém informação usada pelos roteadores para associar um datagrama IP com uma qualidade de serviço pré-requisitada (por exemplo, atraso mínimo)
- ***Payload Length*** (16 bits) - tamanho do datagrama em bits exceto o próprio cabeçalho fixo, cujo tamanho é de 40 bytes. Um datagrama IP pode conter no máximo 2^{16} bits (64 Kb) de dados
- ***Next Header*** (8 bits) - especifica o protocolo do próximo cabeçalho
- ***Hop Limit*** (8 bits) - número máximo de roteadores intermediários que um datagrama pode percorrer antes de ser descartado ou chegar ao seu destino final
- ***Source Address*** (128 bits) - endereço IP de origem
- ***Destination Address*** (128 bits) - endereço IP de destino

6.9.3 Cabeçalhos de extensão e IPSec

Os cabeçalhos de extensão IPv6 são similares às opções do protocolo IPv6. Os cabeçalhos de extensão mais comuns oferecem suporte às funções de fragmentação, *source routing* e segurança. Os cabeçalhos IP de segurança *Authentication Header* (Seção 6.4, página 71) e *Encapsulation Security Protocol* (Seção 6.5, página 73) provêem respectivamente autenticação/ integridade e privacidade de dados.

A Figura 6.12 ilustra um datagrama IPv6 que utiliza os mecanismos de segurança AH e ESP da arquitetura IPSec:

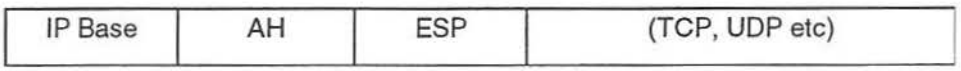

Figura 6.12: IPSec sobre o protocolo IPv6.

O processamento dos mecanismos AH e ESP e protocolos de gerenciamento de chaves associados (SKIP, Photuris etc) segue as mesmas especificações da Arquitetura IPSec definida sobre o protocolo IPv6, conforme abordado no início deste capítulo.

Capítulo 7

Segurança na camada de sessão

Neste capítulo serão apresentados os principais protocolos para obtenção de canais seguros de comunicação no nível de sessão, assim como suas respectivas análises de segurança.

7.1 Introdução

As principais iniciativas de implementação de protocolos na camada de sessão são os protocolos: *Secure Socket Layer* (SSL) [98], desenvolvido pela Netscape, e concorrentemente o protocolo *Private Communication Technology* (PCT) [97], por parte da Microsoft, que são as maiores empresas desenvolvedoras de aplicativos para a Internet.

Ambos os protocolo apresentam APIs de programação proprietárias. Isto implica que as aplicações ficarão dependentes do protocolo de segurança a ser usado. Para evitar tal inconveniente, a IETF (Internet Engineering Task Force) definiu uma interface padrão de acesso a serviços de segurança conhecida como *Generic Security Services* API (GSS-API) [96]. O objetivo principal desta interface é garantir o suporte a múltiplos protocolos de segurança para as aplicações de uma maneira genérica.

Neste Capítulo serão abordados os protocolos SSL e PCT, além da interface GSS-API.

7.2 *Secure Socket Layer* (SSL)

O protocolo SSL é atualmente o mecanismo de segurança mais utilizado na Internet. Seu objetivo é prover uma abstração de canal seguro de comunicação (*stream*), implementando os conceitos de:

- privacidade de dados
- integridade de dados
- autenticação de servidor
- opcionalmente, autenticação de cliente

É importante observar que a propriedade de não repúdio não é oferecida pelo protocolo SSL. Isto representa uma grande lacuna, especialmente no âmbito de aplicações que envolvam transferências de valores.

O nome do protocolo deriva do fato deste implementar uma interface de programação similar às APIs da abstração de *socket* padrão.

A implementação do protocolo HTTP sobre SSL é também conhecida como HTTPS. A Figura 7.1 ilustra esta arquitetura.

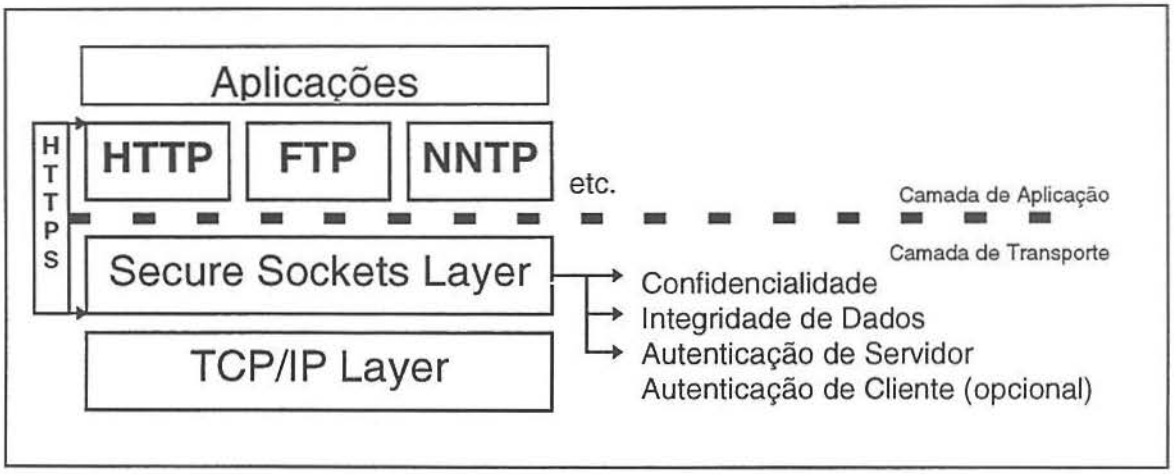

Figura 7.1: Arquitetura HTTPS.

7.2.1 SSLv3.0

A versão 3.0 do protocolo SSL é composta de três níveis lógicos: protocolo de estabelecimento de sessão (*handshake protocol*), camada de transporte (*record layer*) e protocolo de tratamento de erros.

7.2.1.1 Camada de transporte (*record layer*)

A camada de transporte[1] do SSL é usada em todas as comunicações do mesmo, incluindo as mensagens de estabelecimento de sessão (*handshake*), mensagens de erro e transferência de dados das aplicações. De fato, esta camada depende de algum outro protocolo na camada de transporte, como o TCP, para transmissão de dados de maneira confiável.

Os dados a serem transmitidos são fragmentados em blocos de tamanho fixo, opcionalmente comprimidos e finalmente cifrados, conforme definido na transformação de criptografia vigente (estado corrente). Esta transformação compreende o cálculo do MAC (*Message Authentication Code*) do fragmento e o ciframento propriamente dito. Estas operações são determinadas pelo conjunto de algoritmos em uso, denominado de *CypherSpec*, definidos durante a fase de estabelecimento de sessão.

O formato padrão das mensagens SSL é ilustrado a seguir (Figura 7.2):

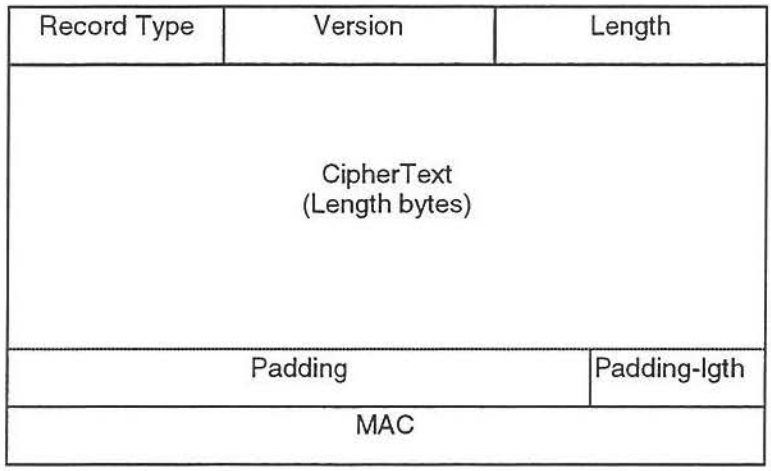

Figura 7.2: Formato de Mensagem SSL.

- ***Record Type*** (1 byte) - Indica um dos seguintes tipos de mensagem: *Handshake, Alert, Change CypherSpec* ou *Data* (dados de protocolos superiores encapsulados)
- ***Version*** (2 bytes) - SSLv2.0 ou SSLv3.0 (atual)
- ***Length*** (2 bytes) - Tamanho do pacote de dados cifrados em bytes

1. Não confundir a camada de transporte do SSL com o conceito de transporte do modelo OSI.

- ***CipherText*** (*Length* bytes) - Dados cifrados
- **MAC** (0, 16 ou 20 bytes, dependendo da função de *hash* em uso) - *Message Authentication Code* dos dados enviados

Caso o algoritmo em uso seja um cifrador de blocos, como por exemplo o DES, pode ser necessário alguma forma de preenchimento (*padding*) para que o tamanho dos dados a serem cifrados seja múltiplo do tamanho do bloco do mesmo. Neste caso, o campo *CipherText* possuirá também os seguintes campos:

- ***Padding*** (*Padding-length* bytes) - preenchimento para que o tamanho dos dados seja múltiplo do tamanho do bloco do algoritmo simétrico em uso
- ***Padding-length*** (1 byte) - tamanho do *padding* usado

7.2.1.2 *Message Authentication Code* (MAC)

O MAC tem a função crucial de oferecer à camada de transporte os princípios de integridade e autenticação. Evita também, no caso específico do SSL, ataques de repetição de mensagens. Para tanto o MAC é calculado através da seguinte fórmula:

hash(MAC_write_secret + pad_2 +

hash(MAC_write_secret + pad_1 + pad_1 + seq_num + length + content)),

onde:

- ***MAC_write_secret*** - é derivado do segredo comum estabelecido (ver Seção 7.2.1.3)
- ***Pad_1*** e ***pad_2*** - *paddings* (preenchimento) fixos
- **Seq_num** - número seqüencial da mensagem em questão
- ***Length*** e ***Content*** - respectivamente o tamanho e conteúdo da mensagem a ser encapsulada
- ***Hash*** - função de *hash* derivada do *CypherSpec* corrente

7.2.1.3 Protocolo de estabelecimento de sessão (*handshake protocol*)

As mensagens SSL são sempre encapsuladas, sobre a camada de transporte SSL, conforme definido pelo *CipherSpec* corrente, ou seja, pelo conjunto de algoritmos de criptografia definidos. No

entanto, o *CipherSpec* inicial é, por *default,* nulo e só será alterado a partir do próprio protocolo de estabelecimento de sessão. Em outras palavras, as mensagens iniciais do protocolo são enviadas em claro, até que um novo *CipherSpec* seja definido.

O estabelecimento de sessão é dividido nas seguintes fases: fase de HELLO, opcionalmente fase de autenticação de cliente, estabelecimento de segredo compartilhado (*master secret*), mudança de *CipherSpec* e finalização. Os parágrafos seguintes descrevem cada fase em maiores detalhes. Em seguida é ilustrado um fluxo de mensagens padrão.

O cliente inicia o protocolo através da mensagem CLIENT_HELLO seguida de um SERVER_HELLO por parte do servidor. Estas mensagens estabelecem os seguintes atributos: versão do protocolo a ser usada (a mais recente comum a ambos), identificador de sessão (*sessionID*), *CipherSuite* (*CipherSpec* mais algoritmo de troca de chaves) e dois valores aleatórios de 28 bytes. Em seguida o servidor envia seu certificado, opcionalmente requer a autenticação do cliente com a mensagem CERTIFICATE _REQUEST, e finaliza esta fase através da mensagem HELLO_DONE.

Se o cliente tiver recebido a mensagem CERTIFICATE_REQUEST, este deve enviar seu certificado público. Um segredo compartilhado é então estabelecido entre o cliente e o servidor através da mensagem CLIENT_KEY_EXCHANGE. O conteúdo desta mensagem depende do algoritmo assimétrico definido durante a fase de HELLO. Se a autenticação de cliente foi requerida, este deve também ser autenticado através da mensagem CERTIFICATE_VERIFY, que é digitalmente assinada com a chave privada deste.

Uma mensagem de mudança do *CipherSpec* (CHANGE_CIPHER_SPEC) é enviada pelo cliente ao servidor. O *CipherSpec* acordado durante a fase de HELLO é finalmente estabelecido. O cliente imediatamente envia uma mensagem de finalização (FINISH), encapsulada pelos novos algoritmos e chaves estabelecidos anteriormente. Em resposta, o servidor também envia as mensagens de mudança de *CypherSpec* e finalização para o cliente. A partir deste momento, cliente e servidor podem transferir dados de aplicações através do canal seguro previamente estabelecido.

A Figura 7.3, página 93, ilustra o fluxo de mensagens no estabelecimento de uma sessão SSL padrão:

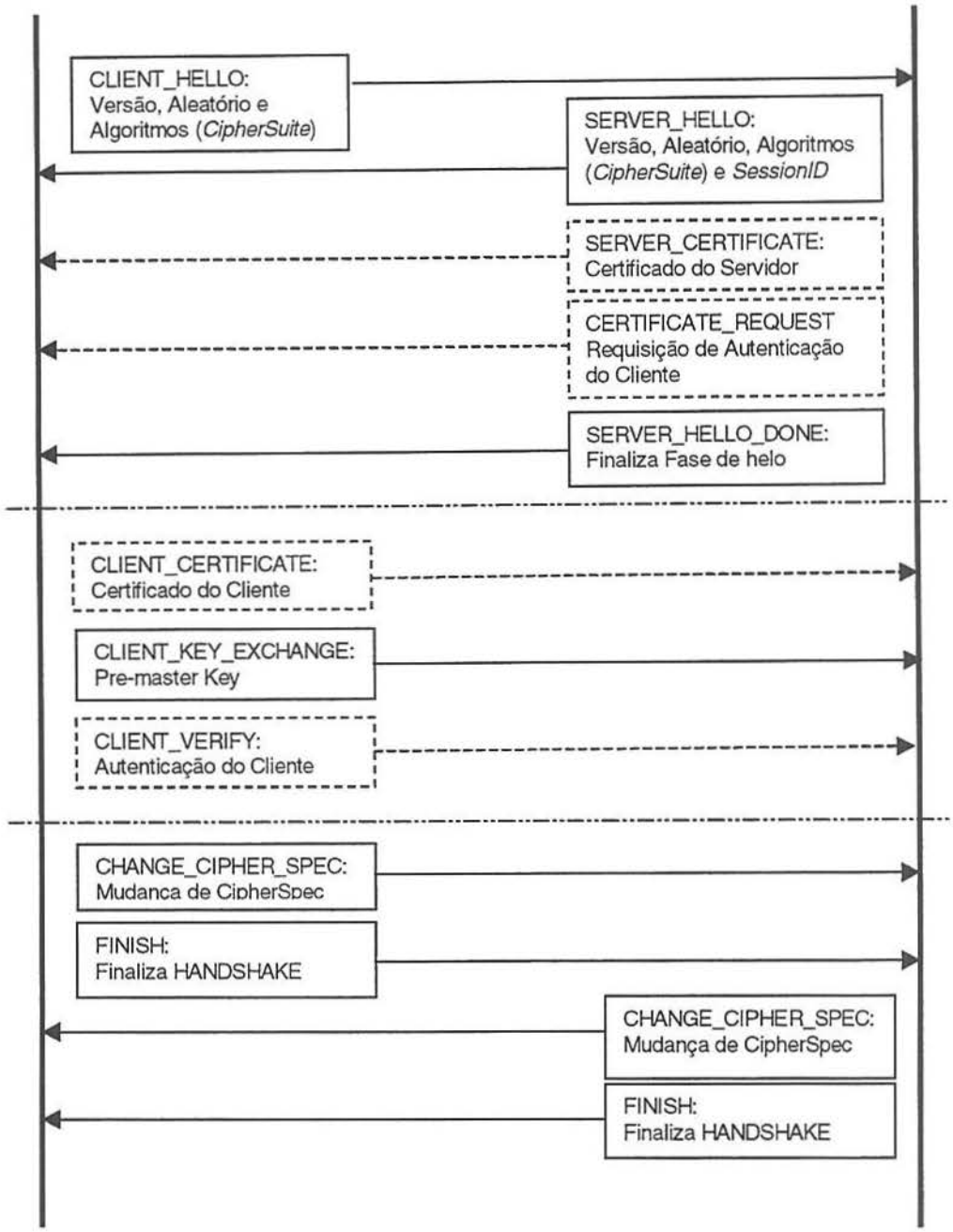

Figura 7.3: Estabelecimento de sessão; as mensagens pontilhadas são opcionais.

7.2.1.4 Restabelecimento de sessão

Caso uma sessão já tenha sido previamente estabelecida, o cliente pode pedir o restabelecimento desta enviando uma mensagem CLIENT_HELLO com o campo *sessionID* identificando a mesma. Para o estabelecimento de novas sessões o *sessionID* deve ser nulo.

O servidor deverá fazer uma busca do *sessionID* recebido em seu *cache* de sessões estabelecidas. Caso o resultado da busca seja positivo, o servidor pode optar por aceitar o pedido de restabelecimento de sessão. Neste caso, o servidor retornará ao cliente uma mensagem de SERVER_HELLO com o mesmo valor do *sessionID* recebido. Em seguida ambas as partes devem prosseguir para a fase de finalização de estabelecimento de sessão, enviando mensagens de CHANGE_CIPHER_SPEC e FINISH.

7.2.1.5 Mensagens SSL

Segue abaixo uma descrição resumida das principais mensagens SSL.

Client Hello

- ***ClientVersion*** - versão com a qual o cliente deseja se comunicar; deve ser a mais recente suportada pelo mesmo
- ***Timestamp*** - data/hora da geração da mensagem
- ***Random*** - aleatório de 28 bytes gerado pelo cliente
- **SessionID** (opcional) - identificador da sessão corrente, caso uma já tenha sido estabelecida previamente
- ***CipherSuite*** - algoritmo de troca de chaves e lista de *CipherSpec* (conjunto de algoritmos de criptografia) suportados pelo cliente; em outras palavras, algoritmos de troca de chaves, autenticação, ciframento e cálculo de MACs

Server Hello

- ***ServerVersion*** - deve ser a maior suportada pelo servidor que seja menor ou igual à requisitada pelo cliente (*clientVersion*)
- ***Timestamp*** - data/hora da geração da mensagem
- ***Random*** - aleatório de 28 bytes gerado pelo servidor

- **SessionID** - identificador da sessão corrente; se o *sessionID* enviado pelo cliente for não nulo e o mesmo for encontrado no *cache* do servidor, este pode determinar o prosseguimento da sessão (*resumed session*) através dos parâmetros anteriormente definidos para tal sessão; caso contrário, um novo *sessionID* é definido
- ***CipherSuite*** - algoritmo de troca de chaves e lista de *CipherSpec* (conjunto de algoritmos de criptografia) suportados pelo cliente

Server and Client Certificate

Certificado público do servidor. Deve seguir o padrão definido pelo *CipherSpec* escolhido. Tipicamente é um certificado X.509v3.

Client Key Exchange

O conteúdo desta mensagem depende do algoritmo de troca de chave escolhido. Tipicamente um segredo compartilhado (*premaster key*) é gerado aleatoriamente no cliente e transmitido cifrado pela chave pública do servidor; ou, através de mecanismos específicos de estabelecimento de chaves, tais como o mecanismo *Diffie-Hellman Key Agreement Protocol* (ver Seção 5.3, página 56).

Certificate Verify

Esta mensagem é usada explicitamente para autenticação de clientes. Após enviar seu certificado público, o cliente, quando requisitada sua autenticação junto ao servidor, deve autenticar-se através de sua chave privada. Assim, uma assinatura digital é produzida pelo cliente, através de sua chave privada, sobre todas as mensagens da fase de *Handshake* recebidas e enviadas até o presente momento.

Finished

Esta é a primeira mensagem enviada cifrada através dos algoritmos e chaves anteriormente estabelecidos. Como as mensagens anteriores da fase de *Handshake* são enviadas em claro, um atacante pode alterar ou substituir tais mensagens com o intuito de influenciar na escolha dos algoritmos a serem usados. Provavelmente, por algum algoritmo que este tenha maior facilidade em atacar.

Além de verificar se o processo de autenticação e troca de chave ocorreu corretamente, a mensagem FINISH têm por objetivo detectar o ataque descrito acima. Isto é feito calculando o *hash* de todas as mensagens da fase de *Handshake*, conforme descrito abaixo, e comparando-o com o valor recebido da outra parte (cliente ou servidor) na própria mensagem FINISH:

hash(master_secret + pad2 + hash(handshake_messages + sender + master_secret + pad1))

Desta forma, caso alguma mensagem da fase de *Handshake* seja alterada, as partes envolvidas serão capazes de identificar a alteração e assim recusar o estabelecimento da sessão em questão.

7.2.1.6 Tratamento de erros

O protocolo SSL possui uma camada específica, conhecida como *Alert Protocol*, para o tratamento de erros e finalização de sessão. Existem dois tipos básicos de mensagens de erros: *warnings* e *fatal*.

Mensagens do tipo *fatal* (fatal) implicam na imediata finalização da sessão corrente. Todos os parâmetros relacionados à sessão corrente devem ser eliminados dos *caches* do cliente e do servidor. A Tabela 7.1 ilustra as principais mensagens do protocolo de erros (*Alert Protocol*):

mensagem	tipo	descrição
close_notify	closure	Finaliza sessão
unexpected_message	fatal	Mensagem inesperada (fora de ordem) é recebida
bad_record_mac	fatal	MAC incorreto. Mensagem adulterada
handshake_failure	fatal	Destinatário foi incapaz de negociar parâmetros de segurança requeridos pelo rementente
no_certificate	warning	Destinatário não possui certificado
bad_certificate	fatal	Certificado corrompido, assinatura não foi verificada corretamente

Tabela 7.1: Mensagens de Erro.

7.2.1.7 Algoritmos oferecidos

Como já mencionado, o SSL é independente de algoritmos específicos. Estes podem ser negociados durante a fase de HELLO através dos *CipherSuite* oferecidos. Um *CipherSuite* define um algoritmo de troca de chaves e *CipherSpec* correspondente (demais algoritmos de criptografia).

Para o algoritmo de troca de chave baseado no RSA, os seguintes *CipherSpecs* encontram-se padronizados e codificados:

CipherSuite SSL_RSA_WITH_NULL_MD5
CipherSuite SSL_RSA_WITH_NULL_SHA
CipherSuite SSL_RSA_EXPORT_WITH_RC4_40_MD5
CipherSuite SSL_RSA_WITH_RC4_128_MD5
CipherSuite SSL_RSA_WITH_RC4_128_SHA
CipherSuite SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
CipherSuite SSL_RSA_WITH_IDEA_CBC_SHA
CipherSuite SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
CipherSuite SSL_RSA_WITH_DES_CBC_SHA
CipherSuite SSL_RSA_WITH_3DES_EDE_CBC_SHA

Note que novos algoritmos podem ser agregados ao protocolo.

7.2.1.8 Chaves de sessão

Durante a fase de HANDSHAKE (mensagem CLIENT_KEY_EXCHANGE) um segredo compartilhado, denominado *premaster_secret*, é estabelecido entre o cliente e o servidor através do algoritmo de troca de chave escolhido. A partir deste segredo inicial é derivado um "segredo mestre" (*master secret*) do qual são derivadas as chaves de sessão e os segredos para o cálculo do MAC das mensagens.

Finalmente, o seguinte conjunto de chaves é gerado: *client_write_MAC_secret*, *server_write_MAC_secret*, *client_write_key*, *server_write_key*, *client_write_IV*, *server_write_IV*. Como existe uma chave para escrita e outra para leitura a chave *server_write_key* equivale a *client_read_key*, e *server_read_key* equivale a *client_write_key*.

A Figura 7.4, página 98, ilustra o fluxo de geração de chaves e segredos. Abaixo, são especificadas as principais fórmulas de geração destas:

master_secret =

MD5(pre_master_secret + SHA('A' + pre_master_secret +
ClientHello.random + ServerHello.random)) +

MD5(pre_master_secret + SHA('BB' + pre_master_secret +
ClientHello.random + ServerHello.random)) +

MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
ClientHello.random + ServerHello.random))

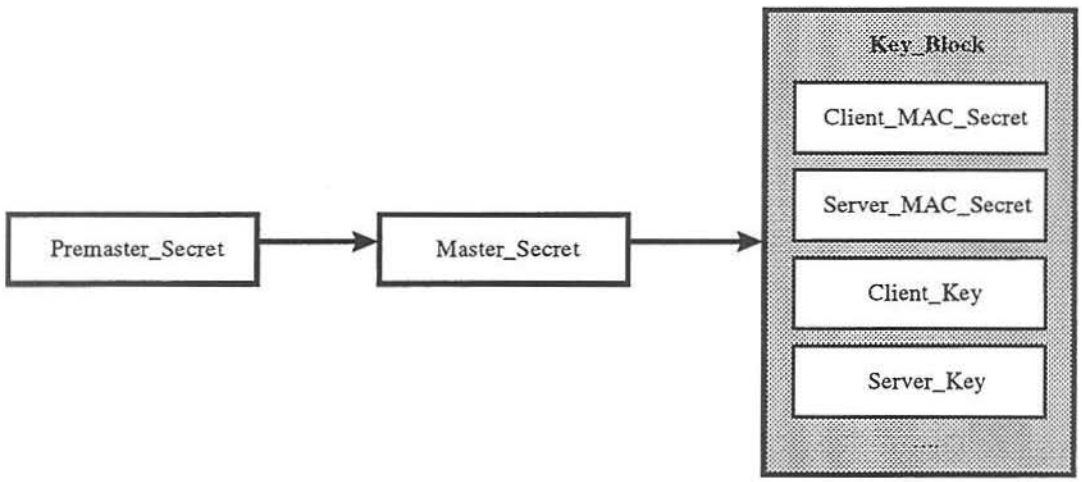

Figura 7.4: Fluxo de Geração de Chaves.

onde *ClientHello.random* e *ServerHello.random* são aleatórios enviados através das mensagens de HELLO.

Em seguida, os segredos MAC e chaves de sessão são seqüencialmente extraídos a partir da variável *key_block:*

```
key_block =

    MD5(master_secret + SHA('A' + master_secret + ServerHello.random +
        ClientHello.random)) +

    MD5(master_secret + SHA('BB' + master_secret + ServerHello.random +
        ClientHello.random)) +

    MD5(master_secret + SHA('CCC' + master_secret + ServerHello.random +
        ClientHello.random)) +

    ... (continua até que todo o Key_Block seja produzido)
```

7.2.2 Vulnerabilidades

7.2.2.1 Interface com aplicação

Mesmo sendo previsto pela especificação do SSL, aplicações que não possuam uma interface de comunicação direta com o protocolo SSL não são capazes de autenticar requisições de clientes, através dos mecanismos previstos na especificação do mesmo (assinatura digital e certificados públicos). Isto ocorre particularmente para aplicações desenvolvidas sobre o protocolo HTTPS.

Além disso, a aplicação não dispõe de mecanismos para identificar ataques relacionados ao canal seguro de comunicação, como por exemplo repetição de mensagens, dentre outros ataques mais sofisticados. Vale notar que a capacidade de reconhecer ataques é de extrema validade para se identificar e prevenir vulnerabilidades em um sistema.

7.2.2.2 *Server and link spoofing*

Na arquitetura SSL a autenticação de servidores é efetuada através de certificados públicos emitidos por entidades certificadoras (*Certificate Authority* - CA), cujos próprios certificados encontram-se estaticamente armazenados nas aplicações clientes (*browsers*). Este esquema oferece uma maneira segura de associar uma URL, por exemplo https://wwws.companiax.com, ao seu respectivo servidor seguro. Isto se o usuário explicitamente abrir a URL em questão. Neste caso, um canal seguro de comunicação é estabelecido entre o servidor citado na URL e o usuário que a referenciou.

No entanto, uma prática extremamente comum na Internet é "navegar" de referência em referência (*link*) até uma página transacional segura obtida através do protocolo HTTPS. Como a página que referencia o *link* seguro é transferida através do protocolo HTTP, esta está sujeita às vulnerabilidades do protocolo TCP/IP. Ou seja, a referência ao *link* seguro original pode ser alterada em trânsito ou uma falsa referência ser fornecida através de técnicas de personificação de máquinas descritas em [66]. Assim, um usuário ao acessar a referência adulterada (possivelmente também uma referência HTTPS) pode ser conduzido a uma outra página, provavelmente em outro servidor e possivelmente com o mesmo aspecto da referência original.

Se a referência adulterada for também uma referência HTTPS, então o indicador de conexão segura nos *browsers* comerciais mostrar-se-á como o esperado. Neste caso, a única proteção oferecida ao usuário é a visualização dos atributos de segurança (certificado SSL do servidor) da URL em questão, o que constitui prática pouco comum entre a maioria dos usuários.

Vejamos então um cenário de ataque: em uma página institucional insegura (http://companhiax.com.br, por exemplo) aparece o link para a "página transacional segura", digamos, https://wwws.companhiax.com.br/scripts/internet2.dll?Pag=Login. Como a página institucional é insegura, o *link* anterior poderia ser substituído para, digamos, https://wwws.ciax.com.br/scripts/internet2.dll?Pag=Login. Obviamente, o intruso teria que registrar um domínio (no caso, wwws.ciax.com.br) para tal fim[1]. Ao entrar no *link* transacional falso, o usuário poderia enviar,

por exemplo o número de seu cartão de crédito inadvertidamente. Após capturar o segredo em questão, o servidor atacante poderia simplesmente interromper a conexão simulando uma falha de comunicação.

7.3 *Private Communication Technology* (PCT)

O protocolo PCT (*Private Communication Technology*) propõe conceitualmente os mesmos princípios de segurança que seu "concorrente" SSL. Ou seja, privacidade, autenticação de servidor, opcionalmente autenticação do cliente.

No entanto, o protocolo PCT é relativamente mais flexível que o protocolo SSL, além de apresentar algumas características adicionais em relação a este. A seção seguinte resume as principais diferenças e semelhanças entre ambos, e em seguida serão descritas as principais peculiaridades do protocolo PCT.

7.3.1 PCT x SSL

O protocolo PCT usa, por questões de compatibilidade, o mesmo formato básico de mensagens do SSL. Servidores que implementem ambos os protocolos devem distinguir clientes SSL de clientes PCT através do número de versão definido no campo *version* (ver Seção 2.1).

A despeito da semelhança em relação à camada de transporte, o protocolo PCT difere em relação ao protocolo SSL, primordialmente em sua fase de HANDSHAKE, principalmente no que se refere aos seguintes aspectos:

- O número e estrutura das mensagens é consideravelmente menor e mais simples. Qualquer pedido completo de estabelecimento de sessão requer somente duas mensagens em cada direção (cliente/servidor). Além disso, um pedido de reconexão requer apenas uma mensagem em cada direção.
- A negociação de algoritmos é mais flexível.

O protocolo PCT oferece também uma série de mensagens especiais para gerenciamento de chaves, suporte a datagramas e mecanismos de recuperação de chaves (*key escrow*). Estas características serão apresentadas em maiores detalhes nas seções seguintes.

---

1. Cabe aos órgãos responsáveis na Internet o zelo com relação ao registro de nomes, principalmente os similares. No Brasil, a FAPESP passou há algum tempo exigir CGC para o registro de nomes no domínio `com.br`.

7.3.2 Handshake protocol

Como no protocolo SSL, durante a fase de HANDSHAKE ocorre a negociação dos algoritmos de criptografia, o estabelecimento de segredo compartilhado, a autenticação do servidor e opcionalmente do cliente. Esta fase consiste de cinco mensagens: CLIENT_HELLO, SERVER_HELLO, CLIENT _MASTER_KEY, SEVER_VERIFY, e CLIENT_VERIFY. Entretanto, apenas as duas primeiras mensagens são obrigatórias em uma fase de HANDSHAKE completa.

A fase de autenticação é intercalada com a fase de estabelecimento de segredo compartilhado. Isto permite uma redução do número global de mensagens transmitidas. O cliente pode, caso este já possua a chave pública do servidor, enviar na mensagem CLIENT_HELLO uma chave mestra cifrada com a chave pública do mesmo. O servidor, de posse desta chave mestra, pode então computar uma resposta de autenticação na própria mensagem de SERVER_HELLO. Os métodos de troca de chaves disponíveis são os algoritmos: RSA, Diffie-Hellman (ver Seção 5.3, página 56) e FORTEZZA[1].

Note que no caso descrito acima, quando o cliente já possuir o certificado público do servidor e não for requerida autenticação do cliente, apenas uma mensagem em cada sentido é necessária para iniciar uma sessão PCT. Uma descrição completa de todas as mensagens PCT durante a fase de HANDSHAKE pode ser encontrada em [97].

7.3.3 Datagramas

O protocolo PCT oferece suporte a datagramas. Datagramas podem ser usados caso a camada de transporte em uso seja "não confiável" (UDP, por exemplo), ou para implementar transmissão de dados fora do fluxo normal de dados (*out of band data*). Um datagrama é uma mensagem que pode ser decifrada independentemente do fluxo de mensagens anteriores da sessão corrente. Em um datagrama PCT as chaves necessárias para decifrar o datagrama e efetuar verificação de MAC são enviadas cifradas com a chave mestra da sessão corrente no começo do próprio datagrama.

1. Algoritmo proprietário da agência de segurança dos EUA (NSA) similar ao Diffie-Hellman.

7.3.4 Gerenciamento de chaves

O protocolo PCT versão 2.0 suporta um tipo especial de mensagens de gerenciamento de chaves (*Key Management Messages*). Estas mensagens implementam uma série de funções especiais que agregam flexibilidade ao protocolo.

7.3.4.1 Dados pré-cifrados

Clientes e servidores PCT podem cifrar dados com uma chave específica, armazená-los e posteriormente enviá-los durante uma sessão futura. A mensagem KM_TYPE_FIXED_KEY contém a chave com a qual os dados foram previamente cifrados, cifrada com a chave mestra da sessão corrente.

As mensagens de gerenciamento de chaves são processadas com a chave corrente da sessão. No entanto, todas as mensagens de dados posteriores a esta são processadas usando a chave específica enviada na mensagem KM_TYPE_FIXED_KEY. Isto ocorre até que uma outra mensagem de gerenciamento (KM_TYPE_FIXED_KEY ou KM_TYPE_RESUME_KEY) indique que a chave original da sessão corrente seja restabelecida.

7.3.4.2 Restabelecimento da chave de sessão original

Como descrito anteriormente, a chave de uma sessão pode ser alterada temporariamente no decorrer da mesma. A mensagem KM_TYPE_RESUME_KEY, neste caso, é usada para restabelecer a chave de sessão (chave de ciframento e cálculo de MAC) original, negociada durante a fase de HANDSHAKE.

7.3.4.3 Requisição de *handshake*

Tanto o cliente como o servidor podem, a qualquer momento após a fase de HANDSHAKE, requisitar que uma nova fase de HANDSHAKE seja estabelecida para a sessão corrente. O servidor pode, por exemplo, requerer que o cliente seja re-autenticado em um ponto específico da sessão, ou que seja autenticado no decorrer desta. Esta mesma função permite limitar a duração das sessões em função da quantidade de dados transmitidos ou da duração total da mesma.

7.3.5 Recuperação de chaves (*key escrow*)

Um tipo especial de mensagem (RT_ESCROW) é definido para facilitar a recuperação de chaves por um nó intermediário (*man_in_the_middle*) entre a origem e o destino de uma mensagem PCT (um *firewall*, por exemplo). Esta mensagem encapsula a primeira mensagem PCT após a fase de HANDSHAKE, além obviamente das chaves usadas nesta. A mensagem PCT original é repassada inalterada para seu destinatário original, logo que as chaves da sessão sejam recuperadas. Assim, todas as mensagens subseqüentes podem ser lidas pelo *escrower* até que uma nova mensagem de gerenciamento de chave seja transmitida, que deve por sua vez ser também encapsulada pelo mecanismo de recuperação de chaves definido acima.

Existem dois tipos de mensagens de recuperação de chaves (EW_TYPE_MASTER_KEY e EW_TYPE_WRITE_KEY). O primeiro recupera a chave mestra (*master key*) da sessão em questão. O segundo recupera apenas as chaves usadas para cifrar os dados propriamente ditos. Dessa forma, é possível definir o nível de confiança no elemento recuperador de chave. Caso o primeiro tipo de mensagem seja usado, o elemento recuperador de chave será capaz não somente de decifrar mas também de alterar as mensagens, recalculando os MACs das mesmas. Pelo segundo tipo, como apenas as chaves de ciframento são recuperáveis pelo elemento recuperador de chave, este não é capaz de alterar as mensagens em trânsito.

No entanto, o mecanismo de distribuição de chaves e o método de ciframento das mensagens de recuperação de chaves propriamente ditos (RT_ESCROW) não são definidos pela especificação do mesmo. Naturalmente, o próprio protocolo PCT pode ser usado para tal finalidade, ou qualquer outro mecanismo de distribuição de chaves comum à origem da mensagem e ao *escrower*.

7.4 *Generic Security Service* API (GSS-API)

O GSS-API (*Generic Security Service Application Programming Interface*) é uma interface de programação genérica que tem por objetivo oferecer serviços de segurança, através de macro funções de "alto nível" para construção de aplicações e serviços de segurança. A RFC 1508 [99] especifica a interface original e, recentemente, a versão 2.0 foi oficializada através da RFC 2078 [96].

Um dos objetivos desta interface é obter isolamento entre a camada de aplicação e os serviços de segurança propriamente ditos. Desta forma, seria possível trocar os algoritmos de criptografia e autenticação de uma aplicação sem alterar seu código. No entanto, uma estrutura de baixo nível, invisível pela interface, deve ser implementada para cada mecanismo de segurança suportado.

7.4.1 Princípio de operação

Comunicações seguras através da interface GSS-API envolvem quatro etapas: obtenção de credenciais, estabelecimento de contextos de segurança, troca de mensagens seguras e finalização de contexto.

7.4.1.1 Credenciais

Obter credenciais é o primeiro passo a ser seguido por um cliente interessado em enviar mensagens seguras para um dado servidor (serviço). As credenciais oferecem os pré-requisitos necessários para o estabelecimento de contextos entre entidades. Estas são obtidas através da chamada `GSS_Acquire_cred`, sendo o mecanismo de segurança definido pelo parâmetro `mech-type`.

Credenciais são elementos privados capazes de autenticar um determinada entidade. Isto inclui uma chave privada para mecanismos de segurança assimétricos ou credenciais do tipo do protocolo Kerberos para sistemas simétricos. O leitor não deve, no entanto, associar diretamente o conceito mais genérico de credenciais da interface GSS-API com o especificamente definido para o último mecanismo citado.

É de responsabilidade do sistema operacional local manter as credenciais de clientes em segurança. Este é um dos aspectos mais críticos na implementação dos mecanismos de segurança em geral.

7.4.1.2 Contexto de segurança

As chamadas `GSS_Init_sec_context` e `GSS_Accept_sec_context` são invocadas respectivamente por clientes e servidores para requerer e aceitar o estabelecimento de um contexto de segurança. Este conceito é semelhante ao de associações de segurança definidos para a arquitetura IP segura (Seção 6.3.3, página 68). O conceito de contexto de segurança é no entanto genérico, e seus detalhes podem variar para cada mecanismo específico.

Os contextos de segurança são estabelecidos entre entidades usando as credenciais obtidas localmente e através de *tokens* enviados entre clientes e servidores. *Tokens* são elementos de dados transferidos entre chamadas GSS-API, que são divididos em duas categorias. Os *Context-level tokens* são responsáveis pelas funções de estabelecimento e gerenciamento de contexto, tais como autenticação de entidades. *Per-message tokens*, por sua vez, são transferidos durante a fase de troca de mensagens para oferecer os serviços de autenticação, integridade e privacidade às mesmas. Ambos os *tokens* são invisíveis do ponto de vista da interface GSS-API, sendo acessíveis apenas pela implementação de cada mecanismo específico.

O iniciador de um contexto (cliente) pode requerer os seguintes serviços:

- autenticação mútua (*mutual-state flag*)
- garantia de seqüenciamento e rejeição de ataques de repetição de mensagens (*replay-det-req-flag sequence-req-flag*)
- proteção da identidade do Iniciador (*anon_req_flag*)

Os servidores GSS-API examinam os *tokens* enviados pelos clientes e determinam se os pedidos requeridos podem ser atendidos ou não. Em caso afirmativo, informam a disponibilidade dos serviços de segurança requisitados, incluindo os referentes à integridade e privacidade de dados (*conf-avail e integ-avail flags*).

7.4.1.3 Proteção de mensagens

Conforme disponível ao contexto de segurança estabelecido, os serviços de integridade/autenticação e privacidade são oferecidos respectivamente através dos pares de chamadas `GSS_GetMIC/GSS_VerifyMIC` e `GSS_Wrap/GSS_Unwrap`. O último par define automaticamente, além de privacidade, os serviços de autenticação e integridade de dados.

As chamadas `GSS_GetMIC/GSS_Wrap` produzem um *token* por mensagem, que é então enviado à outra entidade envolvida. Os *tokens* recebidos são utilizados pelas funções `GSS_VerifyMIC/GSS_Unwrap` para autenticar/verificar integridade e opcionalmente decifrar as mensagens recebidas. Estas operações são efetuadas sobre a parte de dados do protocolo de comunicação em uso.

7.4.2 Descrição da interface

A descrição detalhada da assinatura das funções, isto é dos parâmetros de entrada e resultados retornados, pode ser obtida da RFC 2078 [96]. A título de ilustração, segue abaixo uma listagem de todas as chamadas GSS-API definidas:

chamada	descrição
GSS_Acquire_cred	acquire credentials for use
GSS_Release_cred	release credentials after use
GSS_Inquire_cred	display information about credentials
GSS_Add_cred	construct credentials incrementally
GSS_Inquire_cred_by_mech	display per-mechanism credential

Tabela 7.2: *Credential management.*

chamada	descrição
GSS_Init_sec_context	initiate outbound security context
GSS_Accept_sec_context	accept inbound security context
GSS_Delete_sec_context	flush context when no longer needed
GSS_Process_context_token	process received control token on context
GSS_Context_time	indicate validity time remaining on context
GSS_Inquire_context	display information about context
GSS_Wrap_size_limit	determine GSS_Wrap token size limit
GSS_Export_sec_context	transfer context to other process
GSS_Import_sec_context	import transferred context

Tabela 7.3: *Context-level calls.*

chamada	descrição
GSS_GetMIC	apply integrity check, receive as token separate from message
GSS_VerifyMIC	validate integrity check token along with message
GSS_Wrap	sign, optionally encrypt, encapsulate
GSS_Unwrap	decapsulate, decrypt if needed, validate integrity check

Tabela 7.4: *Per-message calls.*

chamada	descrição
GSS_Display_status	translate status codes to printable form
GSS_Indicate_mechs	indicate mech_types supported on local
GSS_Compare_name	compare two names for equality
GSS_Display_name	translate name to printable form
GSS_Import_name	convert printable name to normalized form
GSS_Release_name	free storage of normalized-formname
GSS_Release_buffer	free storage of printable name
GSS_Release_OID	free storage of OID object
GSS_Release_OID_set	free storage of OID set object
GSS_Create_empty_OID_set	create empty OID set
GSS_Add_OID_set_member	add member to OID set
GSS_Test_OID_set_member	test if OID is member of OID set
GSS_OID_to_str	display OID as string
GSS_Str_to_OID	construct OID from string
GSS_Inquire_names_for_mech	indicate name types supported by mechanism
GSS_Inquire_mechs_for_name	indicates mechanisms supporting name type
GSS_Canonicalize_name	translate name to per-mechanism form
GSS_Export_name	externalize per-mechanism name
GSS_Duplicate_name	duplicate name object

Tabela 7.5: *Support calls.*

7.4.3 Exemplo

Um cenário será apresentado com o intuito de ilustrar a implementação de mecanismos de segurança através da interface GSS-API. De fato, a ilustração abaixo é baseada em um sistema assimétrico ou de chave pública. No entanto, o mesmo pode ser igualmente apresentado para sistemas simétricos como o Kerberos (ver Seção 5.4, página 57).

O iniciador (cliente) de uma conexão segura obteria sua chave privada através da chamada `GSS_Acquire_cred`. Posteriormente, a chamada `GSS_Init_sec_context` utilizaria, por exemplo, a estrutura do DNSSEC (Seção 8.5, página 125) para adquirir de maneira confiável a chave pública do serviço (servidor) desejado. Através da API `GSS_Init_sec_context` gerar-se-ia uma chave de sessão aleatória, e possivelmente também um seqüenciador para evitar ata-

ques de repetição de mensagens. Em seguida os dados gerados seriam cifrados com a chave pública do servidor e assinados com a chave privada do cliente. Estes dados são enviados ao servidor na forma de um *token* de autenticação.

O *token* recebido seria repassado para a função `GSS_Accept_sec_context`, que em seguida obteria a chave pública do cliente. Assim, a autenticidade do *token* é avaliada antes que as chaves de sessão e seqüenciadores sejam aceitos no contexto de segurança.

Uma vez obtida a chave de sessão, um estado seria automaticamente definido entre o servidor e o cliente. A partir de então, as chamadas `GSS_GetMIC` ou `GSS_Wrap` utilizariam a chave de sessão do contexto para operarem os serviços de segurança sobre os dados das mensagens subseqüentes ao servidor. Procedimento inverso seria efetuado no servidor através das funções `GSS_VerifyMIC` e `GSS_Unwrap` para verificação das mesmas.

7.4.4 Mecanismos suportados

Atualmente, bibliotecas de funções GSS-API oferecem suporte para alguns mecanismos de segurança específicos. O suporte a mecanismos e linguagens distintos implica na definição de outras especificações menos genéricas que a RFC 2078. Estas especificações podem ser classificadas em:

- documentos definindo considerações para linguagens específicas, como por exemplo a RFC 1509, que define as interfaces para linguagem de programação C
- definição de formato de *tokens*, protocolos de comunicação e procedimentos de como implementar os mecanismos de segurança em questão

Para este último item destacam-se as seguintes RFCs e implementações correspondentes:

- GSS-API *Authentication Method for* SOCKS *Version* 5 (RFC 1961) [102]
- *The Simple Public-Key GSS-API Mechanism* (SPKM) (RFC2025) [103]
- *Security Mechanism Independence in* ONC RPC (Draft RFC)

Capítulo 8

Aplicações de segurança

No nível de aplicação serão apresentados protocolos seguros abrangendo tanto aplicações para usuários finais na Internet como o que convencionei chamar de "protocolos estruturais", usualmente de maior interesse ao administrador de redes.

8.1 Introdução

Recentemente inúmeras aplicações foram desenvolvidas para prover segurança a aplicativos específicos na Internet. Obviamente seria impossível descrever ou até mesmo enumerar todas as aplicações hoje existentes. Desta forma, é exposto apenas um exemplo de aplicação para três classes de aplicação: comércio eletrônico através do protocolo SET (*Secure Electronic Transaction*), aplicações de correio eletrônico mais populares e, por último, dois protocolos estruturais. Primeiro uma ferramenta genérica de segurança para acesso remoto e administração de redes locais pelo protocolo SSH (Secure Shell), e finalmente a proposta do DNS Seguro (DNSSec).

8.2 *Secure Electronic Transaction* (SET)

O protocolo SET (Secure Electronic Transaction) é uma aplicação desenvolvida pela VISA e pela Mastercard para garantir a segurança de pagamentos usando cartões de crédito na Internet. Dada a popularidade e universalidade deste tipo de pagamento, o SET tende a consolidar-se como padrão de aplicação de comércio eletrônico na Internet.

As seções seguintes descrevem respectivamente o modelo padrão das mensagens SET, o conceito de *dual signature* e o fluxo transacional básico do protocolo.

8.2.1 Mensagens SET

A estrutura básica das mensagens SET é composta por um corpo cifrado mais um envelope digital (Figura 8.1, página 110). O corpo cifrado contém, na maioria das vezes, além da mensagem cifrada propriamente dita, a assinatura digital e certificado público do remetente da mesma. O envelope eletrônico cifra a chave de sessão, usada para cifrar o corpo da mensagem, com a chave pública do destinatário desta. O envelope também encapsula as informações relativas ao cartão de crédito do remetente.

Figura 8.1: Estrutura básica das mensagens SET.

Se um envelope, contendo a chave de sessão usada para cifrar uma mensagem **M**, for cifrado pela chave pública $\mathbf{K_{Pu}}$ de um destinatário, dizemos que **M** foi encapsulada pela chave pública $\mathbf{K_{Pu}}$ pela seguinte notação: $\mathbf{[M]K_{Pu}}$. Em outras palavras, de posse da chave privada $\mathbf{K_{Pr}}$ correspondente de $\mathbf{K_{Pu}}$, é possível decifrar a chave de sessão usada para cifrar **M**, e conseqüentemente a própria mensagem **M**.

Os algoritmos usados para geração de assinatura digital e ciframento do corpo da mensagem são os algoritmos (ver Capítulo 4): DES (algoritmo simétrico), SHA (função de *hashing*), RSA (algoritmo assimétrico) para geração de assinaturas digitais e encapsulamento do envelope.

8.2.2 *Dual signatures*

O protocolo SET cria uma nova aplicação para o conceito de assinaturas digitais. Com este novo conceito, é possível assinar eletronicamente dois documentos associando um ao outro. Para entender a necessidade de negócio deste novo conceito imaginemos o seguinte cenário: um comprador quer enviar uma proposta de compra a um vendedor e uma autorização de transferência de dinheiro para seu banco, caso sua proposta seja aceita. Naturalmente o comprador não deseja que o banco tenha conhecimento da sua proposta de compra, nem que o vendedor tenha acesso a suas informações de conta corrente. Mais ainda, a oferta de compra deve estar associada à transferência, de maneira que esta somente se realizará caso a proposta seja aceita. No contexto SET, *dual signatures* são usadas para associar mensagens de ordem de compra com instruções de pagamento para a instituição financeira do comprador.

Uma *dual signature* é gerada computando-se o *hash* de ambas as mensagens, concatenando-se os dois resultados, e calculando-se novamente o *hash* do valor concatenado e cifrando-se o *hash* final com a chave privada do assinante. O assinante deve incluir o *hash* da mensagem associada juntamente com a mensagem enviada, para que o destinatário possa verificar a assinatura. Assim, o destinatário de cada mensagem pode verificar uma assinatura dual calculando o *hash* da mensagem recebida, e concatenando-o com o *hash* da mensagem associada recebida. Sobre este resultado é novamente calculando um *hash* que é finalmente comparado com a assinatura dual recebida, anteriormente decifrada com a chave pública do assinante.

Ou seja, dadas as mensagens M1 e M2, suas *dual signature* seriam calculadas da seguinte forma:

$$\text{Dual Signature} = \text{RSA}_{\text{chave privada assinante}}[\text{hash}[\text{hash}[M1] + \text{hash}[M2]]];$$

e a mensagem M1 assinada seria formada pelo conteúdo de M1, hash[M2] e *Dual Signature*.

8.2.3 Fluxo transacional

O protocolo SET define os seguintes protocolos transacionais:

- registro de usuário de cartão (*Cardholder Registration*)
- registro de comerciante (*Merchant Registration*)
- requisição de compra (*Purchase Request*)
- autorização de pagamento (*Payment Authorization*)
- transferência de pagamento (*Payment Capture*)

Por motivos práticos e conceituais apenas as transações de Requisição de Compra e Autorização de Pagamento serão abordadas neste documento. Para uma descrição completa referir-se à especificação formal do protocolo [105].

8.2.3.1 Elementos do protocolo SET

As seguintes entidades compõem o protocolo SET:

- **Carteira Eletrônica** - Software usado pelos usuários de cartões (consumidores) para processar as mensagens SET (*Purchase Request*, por exemplo) e gerenciamento de chaves (geração e armazenamento da chave privada do usuário, e validação e armazenamento de certificados públicos).
- **Servidor de Comércio Eletrônico** - Servidor do comerciante que processa os pedido de compra (*Purchase Request*) dos usuários, e repassa as instruções de pagamento para o Gateway de Pagamento. Posteriormente este servidor requer autorização de pagamento para todas as instruções de pagamentos autorizadas.
- ***Gateway* de Pagamento** - Interface de comunicação do protocolo SET com o legado da instituição financeira do usuário e comerciante (operadora de cartões). Emite autorização de pagamento (*Payment Authorization*) para o comerciante, e sob requisição deste (*Payment Capture*) transfere os valores autorizados da conta corrente do usuário.
- **Entidade de Certificação - CA** - Emite e gerencia os certificados públicos das entidades envolvidas no protocolo SET: Usuário (carteira eletrônica), Comerciante e *Gateway* de Pagamento. Cabe à CA manter uma lista atualizada de certificados revogados - *Certificate Revocation List* (CRL).

8.2.3.2 Requisição de compra (*Purchase Request*)

Uma requisição de compra é invocada após um usuário ter selecionado os produtos e serviços a serem adquiridos, escolhido um cartão de crédito específico e definido a forma de pagamento.

O processo de requisição de compra é iniciado quando a carteira eletrônica do cliente requer uma cópia dos certificados do comerciante e *Gateway* de Pagamento (instituição financeira do cliente). O processo é dividido em quatro passos fundamentais: INITIATE REQUEST, INITIATE RESPONSE, PURCHASE REQUEST, PURCHASE RESPONSE, detalhados a seguir:

1. O usuário de cartão de crédito (*cardholder*) efetua uma compra (escolhe o produto, define a forma de pagamento etc).

2. O programa SET do usuário envia uma mensagem INITIATE REQUEST para o comerciante.

3. O programa SET do comerciante, ao receber a requisição inicial, determina um identificador único **ID** para a mensagem recebida; envia ao usuário uma mensagem INITIATE RESPONSE eletronicamente assinada com sua chave privada. Em anexo o comerciante também envia o seu certificado público e o do *gateway* de pagamento.

4. O carteira eletrônica do usuário recebe os certificados públicos e verifica a validade dos mesmos percorrendo toda a hierarquia de chaves até a chave raiz do sistema. Em seguida, a assinatura da mensagem INITIATE RESPONSE é verificada através do certificado do comerciante.

5. O programa SET do usuário gera as mensagens de ordem de compra (*Order Information* - **OI**) e instruções de pagamento (*Payment Instructions* - **PI**) e associa o identificador de transação (**ID**) recebido às mesmas (**OI** e **PI**). Em seguida gera uma assinatura dual para as mensagens acima.

6. O programa SET do usuário encapsula a mensagem **PI** e a assinatura dual de **PI** com a chave pública do *gateway* de pagamento. Esta mensagem ($\mathbf{[PI]K_{PuG}}$), juntamente com a ordem de compra **OI**, a assinada dual de **OI** e o certificado público do usuário, são final-

mente encapsulados com a chave pública do comerciante $\mathbf{K_{PuC}}$. A mensagem resultante $\mathbf{[[PI]K_{PuG},OI]K_{PuG}}$ é, então, enviada ao comerciante.

7. O programa SET do comerciante desencapsula a mensagem recebida através de sua chave privada. Em seguida verifica o certificado público do usuário e a assinatura dual da ordem de compra **OI**; processa a ordem de compra e repassa a mensagem com as instruções de pagamento ($\mathbf{[PI]K_{PuG}}$) para o *gateway* apropriado.

8. O *gateway* de pagamento, de posse de sua chave privada, abre e processa (verifica a assinatura dual) as instruções de pagamento **PI** do usuário. Caso a operação requerida pelo usuário seja aprovada pela instituição financeira, uma confirmação de autorização de pagamento é enviada ao comerciante.

9. O programa SET do comerciante envia a mensagem PURCHASE RESPONSE eletronicamente assinada com sua chave privada, juntamente com seu certificado público para o cliente. Caso a autorização de pagamento seja efetivada pela instituição financeira em questão (*Payment Gateway*), o comerciante envia os bens adquiridos pelo usuário. Posteriormente o comerciante poderá transferir o valor da compra para sua conta corrente através da transação de transferência de pagamento (*Payment Capture*).

8.2.4 Análise de segurança

O protocolo SET envolve uma estrutura bem mais complexa que um canal virtual seguro em uma arquitetura cliente servidor, como ocorre especificamente para o protocolo SSL (ver Seção 7.2, página 88). Obviamente sua natureza é específica para ambiente de pagamentos via cartão de crédito, ou débito direto em conta corrente. Graças ao fato de todas as mensagens serem eletronicamente assinadas, a propriedade de não repúdio é garantida para todas as transações e partes envolvidas no protocolo.

Talvez a principal vulnerabilidade do protocolo esteja na infra-estrutura requerida de certificados públicos e listas de certificados revogados (CRL) para todas as entidades envolvidas: usuários de cartão de crédito, comerciantes e instituições financeiras. Além disso, a segurança local dos sistemas de cada entidade envolvida deve ser bem avaliada, em especial a dos milhares de

sistemas de usuários. A preocupação principal nestes ambientes é quanto à integridade e segurança das chaves privadas dos usuários.

8.3 Aplicações de correio eletrônico seguro

Dentre as várias aplicações e propostas de padrões para garantir a privacidade, integridade e autenticação de mensagens e arquivos, destacam-se como principais especificações o protocolo PGP – *Pretty Good Privacy*, o PEM – *Privacy Enhanced Mail* e recentemente o S/MIME – *Secure Multipurpose Internet Mail Extensions*.

O PEM foi a primeira tentativa oficial, por parte da IETF – *Internet Engineering Task Force*, de definir um padrão para troca eletrônica segura de mensagens na Internet. As RFCs 1421 a 1424 definem o formato das mensagens PEM e uma estrutura hierárquica de certificação de entidades. Como descrito no Capítulo 5, a estrutura de certificação do PEM é rígida e requer o estabelecimento de uma unidade certificadora raiz para toda a estrutura.

O formato das mensagens PEM é baseado em símbolos texto (ASCII) de 7 bits compatíveis com a RFC 822. Para tanto é definido um sistema de conversão de binário para texto conhecido como base64 [100]. A falta de implementações de referencia e a estrutura rígida de certificação foram provavelmente os principais obstáculos à adoção definitiva deste padrão. No entanto, as RFC 1421 a 1424 foram a base e referência para discussão e implementação de diversos protocolos posteriores. O formato de mensagens PEM é usado em várias outras aplicações, como requisição e envio de certificados X.509 de clientes SSL nos browsers comerciais.

O PGP, antes de uma especificação, foi a primeira aplicação a se popularizar como mecanismo de proteção de mensagens e arquivos. A aplicação gera tanto mensagens em formato binário como texto (ASCII). O PGP apresenta uma estrutura informal de confiança onde os usuários são responsáveis pela troca e validação de certificados. Esta estrutura, conforme também descrita em maiores detalhes no Capítulo 5, torna-se ineficiente em ambientes globais como a Internet. Apesar disso, devido à sua flexibilidade, continua sendo uma das ferramentas mais utilizadas de seu gênero.

8.3.1 S/MIME - *Secure Multipurpose Internet Mail Extensions*

O S/MIME tem se popularizado como padrão para composição de um sistema de troca de mensagens seguro. O S/MIME utiliza-se da infra-estrutura dos sistemas de correio eletrônico existentes, e agrega segurança pelo mecanismo MIME (extensão do padrão SMTP).

As características mais relevantes do padrão S/MIME são:

- Estrutura de certificação hierárquica, porém bem mais flexível que no padrão PEM (não é requerida uma entidade de certificação raiz para toda a estrutura).
- Manter a identidade do remetente anônimo. O S/MIME oferece esta característica calculando a assinatura digital da mensagem em primeira instância, e posteriormente encapsulando a mensagem original em um envelope digital com a chave privada do destinatário.

8.3.2 Estrutura de mensagens e certificação

O S/MIME utiliza o padrão de mensagens da PKCS#7 [67], e o padrão PKCS#10 [68] para requisitar e enviar certificados digitais. A PKCS#7 define um formato de mensagem flexível e extensível para representar transformações de dados de criptografia, em especial certificados, assinaturas digitais e dados cifrados. Já a PKCS#10 especifica uma estrutura de mensagens para requisição e envio de certificados X.509.

Várias estruturas de certificação podem coexistir entre si. Normalmente quando uma mensagem é enviada, toda a cadeia de certificação de uma determinada estrutura é também transmitida no formato PKCS#7. Ou seja, todos os certificados desde o remetente até a raiz da estrutura são anexados à mensagem e envelope originais.

Isto aumenta as chances que o destinatário de uma mensagem possa estabelecer um caminho de confiança na estrutura da chave pública do remetente. A inclusão de certificados nas mensagens enviadas pode ser omitida, se os objetos S/MIME forem enviados para um grupo de entidades que tenham acesso ao certificado de cada um dos correspondentes. Neste caso, os agentes S/MIME devem ser capazes de tratar mensagens sem certificados consultando algum serviço público de diretório ou base de dados local.

Estes Agentes locais devem também ser capazes de incluir certificados de CAs, ou seja, certificados auto assinados que constituem a raiz de uma estrutura de certificação. No entanto, mecanismos externos, como a intervenção manual dos usuários, devem existir para determinar se o certificado de uma CA deve ser aceito ou não.

8.3.3 Certificados S/MIME

O padrão S/MIME suporta certificados no formato X.509v1 e X.509v3. Os campos *subject* e *issuer* são definidos pelo formato de *Distinguished Names*, ao invés do endereço eletrônico. Porém, os certificados devem incluir também o endereço de correio eletrônico da entidade certificada, conforme padrão da RFC 822, no campo *subjectAltName* em uma estrutura X.509 estendida.

Listas de revogação de certificados (CRL - *Certificate Revocation List*) devem também ser suportadas nas implementações padrão. Todos os certificados devem ser validados contra CRLs obtidas das autoridades de certificação nas suas respectivas estruturas.

8.4 *Secure SHell* (SSH)

A aplicação SSH permite a execução segura de comandos entre máquinas (comandos "r" de Berkeley: rsh, rlogin, rcp e rdist). Na verdade, o SSH pode ser usado como mecanismo de transporte seguro genérico na camada de aplicação, oferecendo autenticação de máquinas e usuários, além de privacidade e integridade do canal de comunicação. Possui também suporte à propagação (*forwarding*) de aplicações X11 e conexões TCP/IP genéricas sobre o canal SSH seguro.

O protocolo SSH permite completa negociação dos algoritmos a serem usados: algoritmo simétrico, assimétrico, cálculo de MAC, troca de chaves, algoritmos de compressão e formato de chaves públicas e certificados. Tais parâmetros são definidos de maneira transparente para cada sentido de comunicação durante a fase inicial de troca de chaves com o servidor SSH.

Apesar da histórica relação do protocolo SSH com os comandos "r" de Berkeley, este trabalho refere-se essencialmente ao aspecto de segurança do SSH; detalhes relacionados à implementação dos comandos propriamente ditos serão aqui omitidos.

8.4.1 O protocolo SSH

A versão 2.0 do protocolo SSH apresenta as seguintes fases: **identificação de versão**, **troca de chaves e autenticação do servidor**, **autenticação do cliente** (usuário) e **estabelecimento de sessão interativa** [106] [107] [108] [109] [110]. O protocolo consiste de três camadas básicas: camada de transporte, camada de autenticação de cliente e de conexão. A camada de transporte oferece autenticação de servidor, privacidade e integridade de dados, além de propagação perfeita de segredo. A camada de autenticação define um protocolo que implementa vários métodos

de autenticação de clientes sobre a camada de transporte. Finalmente, a camada de conexão multiplexa os vários canais de comunicação de propósito geral.

8.4.1.1 Identificação de versão

Uma conexão SSH é sempre iniciada pelo cliente. O servidor espera por conexões em uma porta TCP fixa (porta 22). Quando o cliente se conecta ao servidor, este responde enviando uma *string* de identificação de versão do tipo "SSH-<versão>.<comentários>". O propósito desta *string* de identificação é validar se a conexão original foi enviada para a porta correta (22) do servidor SSH, e definir a versão do protocolo em uso por cada parte (cliente e servidor).

8.4.1.2 Troca de chaves e autenticação do servidor

Ao contrário das versões anteriores à versão 2.0, o SSH define um mecanismo genérico para troca de chaves. Ao invés de um algoritmo fixo (RSA com duplo ciframento), a versão atual permite ampla negociação dos algoritmos e mecanismos a serem utilizados. O procedimento de troca de chaves *default* é baseado no algoritmo de Diffie-Helman, e a autenticação do servidor em um algoritmo de assinatura digital (RSA-SHA1, DSS etc) anteriormente negociado.

De fato, esta fase se inicia após a troca de mensagens de identificação de versão, com cada lado enviando uma lista completa de todos os algoritmos suportados, organizados em ordem decrescente de preferência. Esta mensagem (SSH-MSG-KEYINIT) é composta basicamente das seguintes informações:

- ***Cookie*** - Valor aleatório de 16 bytes, cujo principal objetivo é possibilitar que ambas as partes (C/S) influenciem na geração da chave e identificador de sessão e evitar ataques de repetição de mensagens.
- **Lista de Algoritmos Suportados** - Algoritmo de troca de chaves, algoritmo de chave pública (assimétrico) para autenticação do servidor, algoritmo de ciframento (simétrico), MAC e compressão do canal de comunicação.

Através do mecanismo de troca de chaves escolhido (DH, por exemplo) um segredo compartilhado $\boldsymbol{K}$ é estabelecido entre o cliente e o servidor. Posteriormente, um identificador de sessão (sessionID - H) é derivado do resultado do *hash* do segredo compartilhado $\boldsymbol{K}$, concatenado com todas as mensagens enviadas anteriormente (identificador de versão e lista de algoritmos suportados).

O identificador de sessão (*sessionID*) é calculado pela seguinte fórmula:

$$H = \text{hash [mensagens anteriores, segredo compartilhado } K]$$

O identificador de sessão ***H*** é então utilizado pelo método de autenticação de servidor escolhido (algoritmo de chave pública do servidor), como dado de entrada para a geração da assinatura digital com a chave privada do servidor. Além de autenticar o servidor, este procedimento garante também a integridade das mensagens da fase inicial do protocolo que são enviadas em claro (identificação de versão e negociação de algoritmos), e evita ataques de repetição de mensagens.

As chaves de ciframento, de cálculo de MAC e os vetores de inicialização (quando o método de ciframento escolhido for em modo CBC) são computados pelo hash do segredo compartilhado **K** e identificador de sessão **H** (sessionID), concatenados a valores pré-definidos.

8.4.1.3 Autenticação de clientes

Após a fase de troca de chaves, o cliente se identifica através de uma mensagem do tipo SSH-CMSH-USER enviando seu *user name*. O servidor sempre responde, durante esta fase, com uma mensagem de sucesso (SSH-SMSG-SUCCESS), caso o cliente tenha se autenticado, ou pelo contrário com uma mensagem de falha (SSH-SMSG-FAILURE). O processo de autenticação é finalizado quando o servidor responder com uma mensagem de sucesso, ou quando o tempo máximo para tentativa de autenticação (*timeout*) tiver se esgotado.

Os métodos de autenticação de cliente suportados são:

- mecanismos tradicionais de autenticação dos comandos "r" através de entradas de usuários e máquinas nos arquivos `.rhost` e `/etc/hosts.equiv` (sistemas UNIX de Berkeley)
- autenticação baseada em assinaturas digitais
- senhas convencionais
- usuários através de entradas no arquivo `.rhost` combinado com autenticação de máquina baseada em assinaturas digitais

Estes métodos serão melhor detalhados na Seção 8.4.3, página 121, que descreve a camada de autenticação do protocolo SSH.

8.4.1.4 Estabelecimento de sessão interativa

Terminada a fase de autenticação, o cliente inicia uma série de requisições para preparar o estabelecimento de uma sessão interativa. Tipicamente estas requisições constituem pedidos de alocação de pseudo terminais (pty), inicialização de aplicações de propagação de serviços X11 e conexões TCP/IP, e execução de um *shell* remoto ou de um comando específico (rcp, rdist etc).

Quando um comando ou *shell* remoto é executado, a conexão passa para o modo interativo. Neste modo, os dados são transmitidos em ambas as direções, sendo que novas conexões podem, então, ser abertas. Como fora anteriormente abordado, uma descrição mais detalhada da camada de conexão foge ao escopo deste trabalho.

8.4.2 Camada de transporte

O protocolo SSH está estruturado sobre uma abstração de mecanismo de transporte própria baseada em pacotes binários.

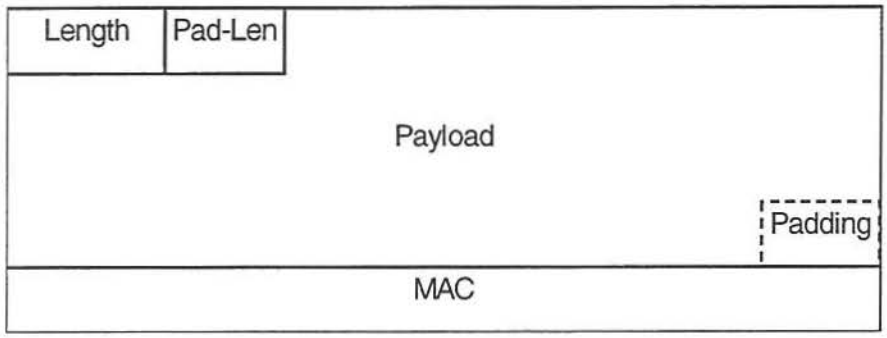

Figura 8.2: Formato de mensagem SSH.

- ***Packet_length*** (2 bytes - *unsigned integer*) - tamanho do pacote de dados em bytes incluindo o *Padding_length* e *Padding*
- ***Padding_length*** (1 byte) - tamanho do *Padding* para que o pacote de dados seja múltiplo do tamanho do bloco do algoritmo em uso
- ***Payload*** (*Packet_Length – Padding_Length* - 1 byte) - dados propriamente ditos
- ***Padding*** (*Padding_Length* bytes) - *padding* para o cifrador de blocos escolhido
- **MAC** (0, 16 ou 20 bytes, dependendo da função de *hash* em uso) - Message Authentication Code dos dados enviados

8.4.3 Camada de autenticação

O SSH implementa um protocolo específico para autenticação de usuários. Este protocolo é construído sobre o próprio canal de transporte seguro do SSH, onde é requerido, autenticação do servidor, e o estabelecimento de um identificador de sessão segura que ofereça também privacidade, integridade de dados e propagação perfeita de segredos aos métodos de autenticação de cliente escolhidos.

8.4.3.1 Protocolo de autenticação

O processo de autenticação é definido pelo servidor, que informa ao cliente quais métodos de autenticação são necessários para acessar um determinado tipo de serviço. O cliente tem a opção de tentar todos os métodos de autenticação oferecidos pelo servidor, na ordem que lhe for mais conveniente.

O processo de autenticação determina um limite máximo de tempo (*timeout*) para aceitar uma tentativa de acesso (10 minutos por *default*). Além disso, um número máximo de tentativas mal sucedidas de autenticação (20 por *default*) deve ser estabelecido. Em ambas as situações, caso os limites definidos tenham sido atingidos, o servidor deve fechar o canal seguro previamente estabelecido.

8.4.3.2 Autenticação baseada em chaves públicas

Neste método o cliente é autenticado enviando uma mensagem de requisição de autenticação (SSH_MSG_USERAUTH_REQUEST) assinada com sua chave privada. O servidor verifica a assinatura recebida com a chave pública associada ao usuário referido. A assinatura gerada pelo cliente é processada sobre os seguintes dados:

- identificador de sessão (*session ID*)
- mensagem de requisição de autenticação propriamente dita

O servidor deve ser capaz de validar as chaves e certificados públicos recebidos. Para facilitar este processo o cliente pode oferecer uma cadeia de certificados digitais, possivelmente até a chave raiz de uma CA que seja de conhecimento do servidor. No entanto, esta verificação pode ser também efetuada através de dados obtidos de uma base de dados local previamente definida.

8.4.3.3 Autenticação baseada em senhas convencionais

Um método tradicional de autenticação baseado em senhas reutilizáveis também é suportado. A maneira como o servidor interpreta a senha recebida e a valida não é definida pelo protocolo SSH. É importante observar que a senha é transmitida sobre o canal seguro estabelecido. Se o canal de comunicação for definido de maneira a não oferecer privacidade (none cipher), este método de autenticação é automaticamente desativado pelo servidor.

8.4.3.4 Autenticação baseada no servidor

Este método de autenticação é similar ao mecanismo definido em sistemas UNIX pelos arquivos `rhost` e `hosts.equiv`, exceto que a autenticação da máquina cliente é efetuada através do método de assinatura digital descrito anteriormente.

Uma vez verificada a identidade da máquina requisitante através de sua assinatura digital, a autenticação do cliente é baseada apenas no *user name* recebido da mesma. Note-se que este método é totalmente dependente do nível segurança e confiabilidade do sistema da máquina cliente.

8.4.4 Transformações de criptografia

Os elementos de transformações de criptografia referem-se aos algoritmos de compressão, de ciframento e de integridade de dados, dos quais as duas primeiras categorias aparecem listadas respectivamente nas Tabelas 8.1 e 8.2.

Algoritmo	Descrição	Recomendação
nenhum	nenhum algoritmo em uso	implementação padrão requerida
zlib	algoritmo Gnu Zlib de compressão de Dados [111]	implementação recomendada

Tabela 8.1: Compressão de dados.

Algoritmo	Descrição	Recomendação
3DES-CBC	DES com ciframento triplo em modo CBC	implementação padrão requerida
Blowfish-CBC	algoritmo Blowfish em modo CBC	implementação recomendada
ArCFor	*stream cipher* compatível com RC4	implementação opcional
IDEA-CBC	IDEA em modo CBC	implementação opcional

Tabela 8.2: Algoritmos de ciframento simétricos; consultar Apêndice A.

Algoritmo	Descrição	Recomendação
CAST128-CBC	CAST-128 em Modo CBC	implementação opcional
nenhum	transmissão em claro	implementação não recomendada

Tabela 8.2: Algoritmos de ciframento simétricos; consultar Apêndice A.

A integridade de dados é obtida calculando-se um MAC (*Message Authentication Code*) para cada pacote binário enviado (Tabela 8.3). O MAC é computado através de uma chave derivada do segredo compartilhado **K**, número seqüencial de 32bits e o conteúdo em claro dos dados enviados, conforme especificado na equação abaixo:

MAC = hash [chave MAC, número seqüencial e pacote em claro]

Algoritmo	Descrição	Recomendação
hmac-sha1	SHA1 saída original de 20 bytes	implementação padrão requerida
hmac-sha-96	primeiros 96 bits do algoritmo SHA1	implementação recomendada
hmac-md5	MD5, saída original de 16 bytes	implementação opcional
hmac-md5-96	primeiros 96 bits do algoritmo MD5 (12 bytes)	implementação opcional
nenhum	nenhum mecanismo MAC aplicado	implementação não recomendada

Tabela 8.3: Algoritmos de integridade de dados (MAC).

8.4.4.1 Método de troca de chaves

De fato, trata-se de um método para estabelecimento de um segredo compartilhado, do qual as demais chaves de sessão são derivadas. Apenas o método baseado no algoritmo de Diffie-Helman (DH Key Exchange) encontra-se atualmente definido (diffie-hellman-group1-sha1).

8.4.4.2 Algoritmo de chave pública (assimétrico)

Os seguintes parâmetros definem o tipo de algoritmo a ser usado.

- formato das chaves e certificados - definem a forma de codificação de chaves e certificados (X509v3, PGP etc)
- assinatura e/ou algoritmo de ciframento assimétrico - alguns algoritmos (DSS, por exemplo) não suportam ciframento de dados, mas somente geração de assinaturas digitais; outros, porém (RSA-SHA), suportam ambas as operações

- codificação de assinaturas e dados cifrados - define padrões tais como: representação de dados (*encoding*), preenchimento (*padding*) e ordem significativa de bits

A maioria dos padrões de certificados e algoritmos de chave pública existentes são suportados pelo SSH (Tabela 8.4):

Algoritmo	Descrição	Recomendação
SSH-DSS	padrão DSS para geração de assinaturas digitais	implementação padrão requerida
X509v3	certificados padrão X509	implementação recomendada
SPKI	certificados SPKI (Public Key Infra-structure)	implementação opcional
PGP	certificados e padrão de assinatura definidos pelo protocolo PGP	implementação opcional

Tabela 8.4: Formato de certificados.

8.4.5 Análise de segurança

O protocolo SSH representa um significativo avanço na segurança de ferramentas de administração remota. A versão 2.0 introduziu uma série de melhorias que tornaram o protocolo mais robusto e confiável, dos quais podemos destacar os ítens que se seguem.

Todos os algoritmos são negociados. Caso algum algoritmo seja cripto-analisado, é possível negociar outro, sem contudo modificar a estrutura geral do protocolo.

As mensagens de negociação inicial de versão e algoritmos, que são transmitidas em claro, são verificadas durante a fase de autenticação do servidor. Como visto anteriormente, as mensagens anteriores a esta fase são assinadas, juntamente com o SessionID da sessão corrente, com a chave privada do servidor. Mecanismo semelhante é também definido pelo protocolo SSL após a fase de negociação de algoritmos (ver Seção 7.2, página 88).

A propagação perfeita de segredo é mantida na versão 2.0. Como o mecanismo de estabelecimento de segredo compartilhado é distinto do mecanismo de autenticação do servidor, os parâmetro do algoritmo de Diffie-Hellman (elemento público) podem variar aleatoriamente para cada nova sessão. Desta forma, se a chave privada do servidor for comprometida, isto não implica na recuperação dos parâmetros usados no cálculo do segredo compartilhado de qualquer sessão anterior, e tampouco das chaves de sessão.

Por outro lado, o protocolo oferece a opção de que a chave pública oferecida por um servidor a um cliente, que esteja se conectando pela primeira vez, seja aceita sem verificação. Apesar

de oferecer flexibilidade na falta de uma infra-estrutura genérica de certificados públicos, esta opção é suscetível à ataques ativos, tais como *man in the middle*.

8.5 DNS *Security Extensions* (DNSSec)

8.5.1 Introdução

Apesar de ser um serviço extremamente importante do ponto de vista de segurança, o DNS está sujeito à diversas vulnerabilidades [15]. Mesmo que sejam corrigidas todas as falhas conhecidas e de implementação, não é possível garantir a segurança do protocolo DNS, principalmente no que se refere às vulnerabilidades intrínsecas da arquitetura TCP/IP convencional.

Em outras palavras, não é possível garantir de maneira totalmente segura a autenticação e integridade das mensagens DNS sem o uso de técnicas de criptografia. Neste contexto, extensões à estrutura do DNS estão sendo propostas [36] com o objetivo de agregar ao protocolo original os seguintes mecanismos de segurança:

- integridade e autenticação de dados através de assinaturas digitais
- serviço de distribuição e gerenciamento de certificados de chaves públicas
- e opcionalmente, autenticação de transações do próprio protocolo DNS, tais como atualizações dinâmicas de registros (*dynamic updates*) [104]

Outro objetivo desta especificação é garantir interoperabilidade com implementações do DNS que não possuam mecanismos de autenticação e geração de assinaturas digitais. Não foram efetuadas, deste modo, mudanças na estrutura das mensagens DNS, além obviamente do acréscimo dos tipos de registros DNS relacionados com os méritos de autenticação e segurança em geral. Desta forma, os serviços acima podem ser suportados por clientes e servidores quaisquer, desde que estes possam tratar os tipos de registros adicionais aqui especificados. Versões superiores da implementação de referência BIND-4.9.4 já apresentam tal compatibilidade.

8.5.2 Verificação de integridade e autenticação de dados

A verificação de integridade e autenticação de dados é obtida associando-se algoritmicamente nomes a seus respectivos registros DNS. Na prática, esta associação é feita com o auxílio de assinaturas digitais.

As assinaturas digitais são armazenadas e recuperadas através de um registro DNS especial, conhecido pelo prefixo SIG (SIG *resource record*). Já as chaves públicas usadas no processo de verificação das assinaturas digitais são tratadas também em registros próprios, conhecidos pelo prefixo KEY (*KEY resource record*). Estes dois novos registros DNS são os principais elementos da extensão de segurança proposta.

Uma zona DNS[1] mantida por um servidor que crie, verifique e distribua registros SIG e KEY é denominada uma **zona segura**. A chave privada usada para gerar as assinaturas de uma zona segura pertencem a esta e não ao servidor DNS que a hospede. Obviamente este pode possuir seu próprio registro KEY com um par de chave pública e privada associado ao mesmo.

8.5.2.1 Registro SIG

A parte de dados específica (RDATA) [29] dos registros SIG inclui o tipo de registro sendo assinado (*type covered*), o algoritmo de criptografia usado para gerar a assinatura digital (*algorithm*), o número de subdomínios do nome associado incluindo o domínio raiz (*labels*), o TTL original (*original TTL*), a data/hora[2] de expiração da assinatura (*signature expiration*), a data/hora que a assinatura foi gerada (*time signed*), um identificador de 16 bits gerado a partir da assinatura (*key footprint* - dependente do algoritmo), o nome (DNS) do assinante (*signer's name*) e a assinatura digital em si (*signature*) codificada em formato base64 [100]. Maiores detalhes sobre os campos descritos acima podem ser encontrados na especificação do DNS seguro [36].

8.5.2.2 Geração e verificação de assinaturas

Para cada registro DNS que se queira obter integridade e autenticação deve-se associar um registro SIG correspondente. Mais especificamente, cada nome de uma zona segura deverá ter associado a este pelo menos um registro SIG para cada tipo de registro existente. Por exemplo, uma máquina com três endereços IP teria um registro SIG para os três registros A associados a esta.

Os dados usados para gerar as assinaturas digitais são obtidos agrupando todos os registros de um mesmo tipo para um determinado nome. Em seguida, estes são então canonicalizados. Isto envolve expandir quaisquer nomes comprimidos, transformar todas as letras maiúsculas em

1. Uma zona DNS é composta apenas pelos registros de responsabilidade (*authoritative*) do servidor desta. Diferentemente de um domínio DNS, os registros dos domínios delegados não fazem parte da zona em si.
2. Representado como número de segundos a partir de primeiro de janeiro de 1970, GMT.

minúsculas, substituir o TTL corrente pelo TTL original e finalmente classificar os registros em ordem crescente por nome. O objetivo da canonicalização é garantir a unicidade dos dados a serem assinados.

No sentido inverso, para verificar a assinatura e conseqüentemente autenticar um dado registro, é preciso recuperar todos os registros deste mesmo tipo associados ao nome em questão, canonicalizá-los e finalmente obter a chave pública da zona à qual estes pertençam. Antes de regerar e conferir a assinatura digital, o registro KEY (chave pública) da zona tratada deve ser igualmente validado. Este processo de validação é descrito em maiores detalhes na seção seguinte.

Com o objetivo de otimizar o processo de geração e validação de assinaturas, servidores de zonas seguras tentarão sempre retornar os SIGs correspondentes aos registros requeridos.

8.5.2.3 Outros registros de segurança

Glue records

Um servidor normalmente envia como resposta, quando consultado sobre um domínio que este tenha delegado autoridade, as seguintes informações: registro NS e *glue record* do servidor delegado (*referral data*), chave pública (KEY) da sub-zona delegada e assinatura digital (SIG) da mesma.

Para evitar conflito e ambigüidade, *glue records* e registros NS devem ser assinados apenas pela zona à qual pertençam. A zona pai deve assinar somente o registro KEY da sub-zona delegada. Note que a chave pública (KEY) de uma zona é um registro cuja autoridade (*authoritative*) pertence à zona pai desta.

Para validar o *glue record* recebido, uma *query* específica deve ser feita ao servidor da sub-zona para obter o conjunto de seus registros A e o SIG correspondente. É interessante observar que esta é uma mensagem adicional em relação ao protocolo DNS convencional.

Nomes e tipos inexistentes

O comunicado de inexistência de um dado tipo de registro ou nome em uma zona não tem como ser facilmente autenticado. Os registros SIG até então definidos podem somente autenticar os registros DNS existentes. O estabelecimento de um mecanismo para indicar a inexistência de um determinado nome ou tipo de registro é crucial para evitar ataques de negação de serviço.

Um tipo especial de registro, conhecido pelo prefixo NXT (*next record*), é usado para definir intervalos a partir dos nomes existentes, ordenados em ordem alfabética. Cada intervalo significa que não existe nenhum nome válido entre os especificados pelo registro NXT. Para garantir autenticidade um registro SIG é associado a cada intervalo definido. Os registros NXT de uma zona devem ser automaticamente calculados e assinados durante o processo de assinatura dos demais registros desta.

Suponhamos que a zona dcc.unicamp.br possua os seguintes nomes de máquinas:

```
grande.dcc.unicamp.br.
nova.dcc.unicamp.br.
velha.dcc.unicamp.br.
```

Uma *query* para o nome `pequena.dcc.unicamp.br` implicaria no seguinte registro NXT e SIG retornado:

```
nova.dcc.unicamp.br. NXT velha.dcc.unicamp.br A SIG NXT
nova.dcc.unicamp.br. SIG NXT 1 3 ( ;tipo-assinado=NXT, alg=1-RSA/SHA,
label=3
     19980710030405 ; signature expiration
     19970710030405 ; signature expiration
     15497375; key footprint
     dcc.unicamp.br; assinante
Umopqiemvcve7/sdjmbc43m9jdjkaklaskleklaioliajvmBieialj/aslijdoiweu//
zmbbIKSlajalkflaksjdflaksjdf=; sig-base64)
```

Os registros indicados (*type bit map*) após o nome *velha.dcc.unicamp.br* (A, SIG e NXT) indicam a existência destes para o nome `nova.dcc.unicamp.br` (*owner name*).

Aliases DNS

Pela especificação atual do DNS, *aliases* devem possuir um único registro CNAME associando-o ao seu nome oficial. *Queries* efetuadas para *aliases* retornam, na verdade, os registros associados ao nome canônico do mesmo. No DNS seguro, no entanto, é necessário ter um registro SIG para cada registro que se queira autenticar. Isto requer uma modificação no comportamento dos servidores DNS, que devem passar a retornar um registro SIG ou NXT inclusive para registros CNAME.

AXFR SIG

O objetivo deste registro é facilitar a verificação de transferência de zonas (*zone transfers*). Desta forma, uma zona inteira, incluindo os próprios registros SIG, pode ser assinada e o resultado armazenado em um único registro AXFR SIG. Após a validação deste, a zona pode ser considerada segura sem a necessidade de se verificar todos os registros SIG nela contidos.

8.5.3 Distribuição de chaves

Inexiste atualmente uma infra-estrutura global, prontamente disponível, capaz de gerenciar e distribuir certificados de chaves públicas para as demais aplicações na Internet (PGP, SKIP etc). Neste cenário, a estrutura de distribuição de chaves do DNS seguro tem sido sugerida como alternativa para a problemática da distribuição de chaves em âmbito genérico na Internet.

A infra-estrutura original do DNS apresenta as seguintes características desejáveis a um sistema de distribuição de chaves:

- plena disponibilidade
- estrutura de nomes única e não ambígua
- acesso em tempo real

Para dar suporte ao mecanismo de autenticação e integridade de dados, o DNS seguro implementa uma estrutura segura para a distribuição de chaves públicas. Mecanismos nativos foram definidos para a distribuição de chaves públicas autenticadas através de registros KEY e SIG correspondentes. O processo de obtenção e validação de chaves públicas é o mesmo que para os demais tipos de dados do DNS. Conseqüentemente, nomes podem ser associados a suas chaves públicas através do DNS seguro.

8.5.3.1 Registro KEY

Os registros KEY foram definidos para associar nomes DNS à(s) chave(s) pública(s) de entidades. A parte de dados específica dos registros KEY inclui um campo de *flags*, o protocolo que vai utilizar a chave pública (*protocol*), o algoritmo gerador (*algorithm identifier*) e a chave pública (*public key*) propriamente dita.

Nomes a serem associados podem representar uma zona, uma máquina, ou o usuário de um sistema. Alguns bits do campo *flags* são usados para determinar a qual das categorias acima um

dado nome pertence. Também através do campo *flags* é possível especificar a inexistência de chave para um determinado nome. Desta forma, uma zona insegura, sem as extensões de segurança, pode ser assim qualificada de maneira formal pelo protocolo.

8.5.3.2 Mapeamento de endereços de correio eletrônico em nomes DNS

O espaço de endereçamento de nomes de usuários usado pela maioria das aplicações na Internet refere-se normalmente a endereços de correio eletrônico. Portanto, é necessário estabelecer algum mecanismo para mapear endereços de correio eletrônico em nomes DNS.

É bem provável que os endereços de correio eletrônico (caixas postais) da maior parte dos sistemas possam ser facilmente convertidos em nomes DNS. Na maioria das instalações, estes são constituídos pelo nome do usuário (*username*), o separador "@", e o nome do domínio em questão. Os caracteres válidos para os nomes DNS são as letras de A a Z, números e o caracter "-". Nomes de usuários que não possuam caracteres especiais, como "." ou "_", podem ser mapeados em nomes DNS simplesmente substituindo o separador "@" por um ponto ".". Por exemplo, duarte@dcc.unicamp.br seria mapeado em duarte.dcc.unicamp.br.

Atualmente, o caracter "\" é usado para representar endereços eletrônicos que possuem "." no registro SOA (*Start of Authority*). Assim a seqüência "\." não é interpretada como separador de domínio [112]. O nome joao.silva@dcc.unicamp.br é representado como joao\.silva.dcc.unicamp.br. Para o caso genérico, regras de mapeamento específicas ainda devem ser definidas.

8.5.3.3 Caminhando de maneira segura pela árvore do DNS

Para validar as chaves públicas de entidades obtidas através do DNS é necessário obter *a priori*, por outro meio seguro, a chave pública de algum servidor na hierarquia DNS. Servidores e clientes DNS precisam ser iniciados estaticamente com este tipo de informação para poderem autenticar, de maneira segura, as chaves públicas obtidas de outras zonas.

A questão é definir com que chave pública de qual servidor um cliente ou servidor local deve ser pré-configurado.

Enfoque "*Top-Down*"

Uma possibilidade é configurar no sistema a chave pública do domínio raiz. Assim é possível descer toda a hierarquia de maneira segura, validando cada chave de sub-domínio recebida com

a do domínio superior a partir da raiz. A chave de uma zona é identificada pelo registro KEY de um NS, fornecido pelo servidor do domínio raiz, e é validada conferindo a assinatura digital do SIG correspondente através da chave pública do domínio superior.

Esta é certamente a maneira mais segura de se configurar um cliente DNS. Descendo pela hierarquia do DNS, servidores só são capazes de "mentir" para nomes que estes sejam responsáveis (*authoritative*). Ainda assim, é possível identificar e atribuir responsabilidade à zona maliciosa. Se as chaves privadas dos domínios superiores forem mantidas em segurança, a confiança no sistema será bastante elevada.

Um pré-requisito à segurança do enfoque acima é que todos os sub-domínios desde a raiz até o destino final sejam seguros[1]. Isto pode não ser verdade, principalmente durante a fase inicial de implantação do sistema. Existem também alguns inconvenientes administrativos. A mudança da chave pública do domínio raiz requer a alteração de milhares, talvez milhões, de clientes e servidores em todo o mundo. Além disso, muitos domínios podem não confiar no domínio raiz.

Enfoque "*Bottom-Up*"

Outro procedimento intuitivo é configurar os sistemas locais com a chave pública do domínio a que pertençam. Os clientes DNS são na maioria das vezes associados a um servidor local, no qual normalmente confiam plenamente.

Um servidor responsável por uma zona segura deve, além de assinar as chaves públicas das sub-zonas que este é responsável (*authoritative*), incluir também um registro KEY (chave pública) de sua zona superior (pai), assinado por ele próprio. O objetivo deste procedimento é permitir que se suba pela árvore DNS de maneira "segura".

A chave pública da zona superior não é um registro *authoritative* da zona delegada. Esta desconexão permite que servidores maliciosos mintam a respeito da porção superior da árvore DNS. Na pior situação, um servidor raiz falso poderia ser indicado, o que comprometeria todo o espaço de endereçamento. Neste aspecto esta topologia é bem mais vulnerável que a anterior.

Um enfoque híbrido seria ter os clientes locais associados estaticamente a seu servidor local, e este à chave pública do servidor raiz e de alguns de seus vizinhos confiáveis. Isto permitiria

1. Seguros no sentido de implementarem as especificações do DNSSEC.

maior flexibilidade para se efetuar mudanças, como por exemplo a troca da chave pública do servidor local. Contudo as *queries* remotas teriam a segurança do enfoque *"Top-Down"*.

8.5.4 Considerações de segurança

8.5.4.1 Privacidade

O objetivo do DNS é disponibilizar informação de maneira global e não discriminatória. Devido a esta filosofia, nenhum mecanismo foi previsto na especificação do DNS seguro para garantir privacidade ou controle de acesso às informações tratadas pelo sistema.

8.5.4.2 Chave privada de uma zona segura

É recomendado que a chave privada usada para assinar os registros de uma zona seja armazenada fora dos servidores da mesma. Mais ainda, que os registros sejam assinados em um dispositivo ou máquina fisicamente seguros e depois transferidos para os servidores DNS propriamente ditos. Assim, mesmo que os servidores de uma zona sejam comprometidos, os registros DNS desta não teriam como ser falsificados. Este procedimento tende a aumentar em muito o grau de confiança do sistema como um todo.

8.5.4.3 Revogação de registros

A especificação do DNS seguro não inclui nenhum mecanismo explícito para revogação de registros. Uma das principais vulnerabilidades dos sistemas de distribuição de chaves baseados em certificados públicos é a falta ou a má administração de tais mecanismos.

No DNS convencional, a alteração do valor de algum registro de um determinado nome, como por exemplo o endereço IP de uma máquina, é propagada com o auxílio do parâmetro TTL (*time to live*), que define o tempo de validade do mesmo. No entanto, este mecanismo requer a cooperação de todos os participantes do sistema, ou seja, todos os servidores devem periodicamente expurgar de seus *caches* registros que estejam com o tempo de validade (TTL) vencido.

Basicamente a mesma idéia é usada no DNS seguro para revogar a validade de registros alterados. Exceto pela proteção oferecida pelo registro SIG, nenhum mecanismo de segurança é acrescentado ao procedimento anterior. A confiança nos vários elementos do sistema continua sendo um fator chave para a atualização da base de dados global.

Na ausência de mecanismos de revogacão mais adequados, como CRL (*Certificate Revocation Lists*), registros antigos, capturados juntamente com seus SIGs correspondentes, podem ser enviados em respostas falsas, sem que isto possa ser identificado. A situação, neste caso, é mais crítica quando o registro a ser revogado for a chave pública de algum domínio.

Capítulo 9

Conclusão

Em oposição à pouca preocupação inicial com o aspecto de segurança na especificação original dos protocolos da arquitetura TCP/IP, existe hoje um espectro bastante variado de soluções de segurança, principalmente em relação aos protocolos de segurança propriamente ditos.

Esta diversidade de soluções é resultado de uma crescente preocupação com o aspecto de segurança da Internet, e demonstra principalmente a viabilidade técnica de soluções para os mais diversos aspectos e propósitos, sejam estes mero controle de acesso através de *firewalls* e demais técnicas apresentadas no Capítulo 3, ou protocolos de segurança capazes de garantir a segurança de aplicações críticas como transações financeiras e comércio eletrônico.

Mesmo com a evolução das técnicas de segurança, as vulnerabilidades levantadas no Capítulo 2 não podem ser consideradas histórico de um passado recente. Serviços básicos de informação na Internet e os protocolos que provêem infra-estrutura básica de roteamento, mapeamento de recurso etc, deverão continuar operando sobre os protocolos clássicos sem proteção de técnicas de criptografia.

Além disso, falhas nos protocolos básicos podem não ser simplesmente resolvidas através uma infra-estrutura segura. Ataques de negação de serviço contra o protocolo TCP, tal como SYN *Flooding*, podem continuar factíveis, mesmo que se esteja operando sobre a arquitetura IP segura (IPSec). Ainda assim, os protocolos seguros podem estar sujeitos a falhas de implementação, justamente no que se refere a aspectos do protocolo em si. As vulnerabilidades nativas da arquitetura TCP/IP devem servir de exemplo para evitar que se cometam os mesmos erros em novas especificações.

Medidas tais como manutenção de estado de conexões, associação de pedidos com respostas e utilização de seqüências aleatórias são práticas necessárias na construção de qualquer tipo de protocolo seguro, seja ele convencional ou que se utilize de técnicas de criptografia.

Contudo, sem a utilização de técnicas de criptografia é impossível, em um ambiente totalmente aberto como a Internet, implementar os conceitos de privacidade, autenticação e integridade de dados. Talvez por isto, a Internet tenha sido o meio que mais difundiu o uso e o desenvolvimento desta técnica nos últimos anos.

Neste cenário, tornou-se indispensável aos desenvolvedores de protocolos e aplicações na Internet o conhecimento básico destas técnicas. O Capítulo 4 procurou apresentar este assunto de maneira intuitiva, pouco formal, mas sem deixar de oferecer o conhecimento mais aprofundado e comparativo necessário à compreensão básica do assunto. Novas técnicas foram apresentadas, com destaque para os sistemas baseados em curvas elípticas e os sistemas assimétricos em geral. O Apêndice A apresenta um quadro resumo dos principais algoritmos simétricos e assimétricos disponíveis ao público em geral.

Juntamente à divulgação dos sistemas de criptografia torna-se necessária a construção de infra-estruturas de gerenciamento de chaves públicas ou *Public Key Infrastructure* (PKI), em especial entidades de certificação (CA). Atualmente as entidades certificadoras são um elemento chave para segurança e divulgação do comércio eletrônico na Internet, mesmo que atuando em caráter não exatamente formal. Em outras palavras, seu uso não foi ainda regulamentado, como acontece com as agências oficiais que administram o registro de domínios DNS.

Os protocolos de segurança que se utilizam de técnicas de criptografia para implementar os conceitos de privacidade, autenticação, integridade e não repúdio foram genericamente classificados conforme sua camada de implementação. Os Capítulos 6 e 7 abordaram os protocolos que criam o conceito de canal seguro de comunicação, ou seja, uma abstração de um meio de comunicação seguro, onde o servidor é autenticado, e a privacidade e integridade do canal de comunicação, preservadas.

O protocolo SSL (*Secure Socket Layer*) é certamente o mais bem sucedido esforço de construção e padronização de um canal seguro de comunicação. Apesar de ter sido uma iniciativa privada, o SSL é de fato um padrão bem mais aceito e difundido que a iniciativa oficial da arquitetura IP segura (IPSec). Atualmente o SSL encontra-se disponível na maioria das aplicações de cliente (*browsers*) e servidores (WWW, FTP, SNMP etc) na Internet.

Teoricamente, a arquitetura IP segura pode ser usada com o mesmo intuito que os protocolos de sessão (SSL, PCT etc) para implementar canais seguros de comunicação, mas de maneira muito mais transparente para as aplicações existentes. Na prática, porém, os protocolos propostos para a camada IP (IPSec) são mais comumente utilizados para suportar redes de perímetro virtual (VPN), atualmente também conhecidas como Extranets seguras.

Além do conceito de canais de segurança, a maioria destes protocolos também oferece autenticação "forte" de cliente através de técnicas de algoritmos assimétricos e certificados públicos. Mesmo assim, poucas aplicações práticas fazem uso desta facilidade atualmente. Ainda hoje, a despeito das vulnerabilidades apresentadas, uma maciça quantidade de aplicações efetua autenticação de clientes através de "senhas reutilizáveis". No Capítulo 7 foi apresentado um exemplo prático de ataque a aplicações desenvolvidas sobre o protocolo SSL, e cuja autenticação era baseada em senhas reutilizáveis.

Finalmente, o Capítulo 8 apresentou uma série de exemplos de aplicações, protocolos estruturais de segurança, e padrões de correio eletrônico seguro. Dentre estas aplicações destacam-se aquelas voltadas ao comércio eletrônico, onde os requisitos de segurança, em especial o não repúdio, são de crucial importância.

O exemplo mais notável de comércio eletrônico é o protocolo SET. Específico para o pagamento de compras via Internet, e contrariamente ao protocolo SSL, o SET não tem propósito genérico. No entanto, este tende a se tornar padrão, substituindo o uso de números de cartão de crédito e o protocolo SSL, para efetuar pagamentos na Internet.

O não repúdio é potencialmente mais difícil de ser implementado que as demais propriedades de segurança anteriormente mencionadas. Enquanto privacidade, integridade e autenticação podem ser providas de maneira transparente para a camada de aplicação, o não repúdio só pode ser oferecido através de uma interface direta da aplicação (API) com o protocolo de segurança e recursos de criptografia.

Na prática, mensagens específicas, como requisições de transferência de valores, devem ser digitalmente assinadas e armazenadas para posterior verificação. Além disso, a chave privada dos clientes deve ser independentemente gerada por estes, e posteriormente emitido um certificado para a chave pública correspondente. Fica evidente que a implementação de não repúdio em aplicações requer um custo adicional no gerenciamento de certificados de clientes, requisições

de assinaturas digitais em transações específicas, e no armazenamento e validação destas mensagens pelo servidor.

Devido às dificuldades apresentadas, o princípio de não repúdio é ainda pouco difundido, com exceção do protocolo SET e das aplicações de correio eletrônico seguro. Recentemente a Netscape disponibilizou mecanismos nativos para efetuar assinatura digital em formulários HTML (*sign forms*) em seu *browser*. Isto abre a possibilidade de implementar não repúdio em aplicações genéricas, baseadas sobretudo, no protocolo SSL sobre HTTP (HTTPS) e linguagem HTML (*Forms*). Desta forma, a preocupação com o conceito de não repúdio deve se intensificar, seja em função do aumento de transações financeiras na Internet ou do aumento da infra-estrutura básica de segurança disponível nos produtos de mercado.

A respeito das aplicações de correio eletrônico seguro vale salientar que o padrão PEM (*Privacy Enhanced Mail*) teve seu maior valor como proposição de uma estrutura de segurança genérica para a Internet. Já o protocolo PGP surgiu de uma aplicação proprietária, mas de grande popularidade, e estrutura de certificação informal, não hierárquica. No entanto, o padrão MIME tem atraído mais interesse ultimamente, em especial devido à sua presença nas versões de *browsers* mais populares.

Complementando a demanda por segurança na Internet, existe uma necessidade de prover mecanismos de segurança para administração de redes, e em especial para a construção de uma infra-estrutura de gerenciamento de chaves públicas e certificados. Sem a ousadia de ser completo, em relação às propostas existentes, os protocolos SSH (*Secure Shell*) e DNSSEC (*Secure* DNS) foram apresentados, completando assim o espectro de aplicações de segurança.

A grande variedade de soluções e protocolos de segurança representa uma resposta positiva em relação às necessidades da Internet. Contudo, esta variedade pode tornar difícil identificar o tipo de aplicação, ou protocolo de segurança, mais adequado a uma determinada necessidade. Novamente, os protocolos de segurança foram analisados e agrupados em função de suas funcionalidades e camada de atuação: nível IP, camada de sessão e aplicação. A tabela abaixo os resume, conforme seu nível de atuação:

Existem, também, inúmeras referências sobre segurança em geral. No entanto, poucas ou nenhuma destas referências abordam os protocolos de segurança, enfocando o modelo seguro da arquitetura TCP/IP. A principal meta deste trabalho foi oferecer, ao usuário e desenvolvedor na Internet uma literatura básica que cobrisse todo o espectro de segurança no que tange à

Descrição	Exemplos	Vantagens	Desvantagens
Camada de Rede			
Implementações de VPNs baseadas, por exemplo, na Arquitetura IP Segura IPSEC.	SKIP Photuris ISAMAP/ Oakley	Padrão RFC (Request for Comments). Transparente para qualquer aplicação. Interoperabilidade entre soluções.	Falta de protocolo padrão para o gerenciamento de chaves.
Camada de Sessão			
Abstração de canal seguro na camada de transporte.	SSL PCT	Transparente para abstração de *streams*. bastante difundido (presente nos principais browsers comerciais).	Deve ser implementado em cada aplicação. Não oferece "não repúdio".
Camada de Aplicação			
Aplicações de correio eletrônico seguro que oferecem privacidade, autenticação, integridade e não repúdio as mensagens.	PEM PGP MIME	PGP: não requer PKI (certificação informal); bastante difundido. PEM: arquitetura de segurança global, incluindo PKI. MIME: mais flexível e tendência de tornar-se padrão.	PGP: não possui estrutura hierárquica de certificação. PEM: complexo e pouco flexível.
Secure Electronic Transaction - SET. Protocolo específico para pagamentos eletrônicos seguros.	SET	Implementa os serviços de segurança básicos de autenticação, recibos digitais e não repúdio, necessários para se efetuar transações seguras.	Específico para pagamentos eletrônicos. Não é de propósito geral. Criptografia fraca: DES (56 bits).
Extensão de segurança do serviço de DNS. Implementa uma PKI (*Public Key Infrastructure*).	DNSSEC	Mecanismo e infra-estrutura já bastante conhecidos e difundidos.	Requer estabelecimento de hierarquia rígida de autoridades de delegação de domínios.
Secure Shell - Abstração Segura dos comandos Shell ("r") remotos em UNIX.	SSH	Administração remota e canais seguros entre máquinas em ambiente UNIX.	Pouca disponibilidade em outras plataformas (Windows, MacOS etc).

Tabela 9.1: Principais protocolos e aplicações de segurança.

obtenção de canais seguros de comunicação. Isto inclui a especificação original do protocolo TCP/IP, vulnerabilidades nativas, conceitos básicos de criptografia, gerenciamento de chaves e os protocolos de segurança propriamente ditos, que são os sustentáculos desta nova era da Internet, a era de transações seguras e do comércio eletrônico.

Bibliografia

[1] *Request for Candidate Nomination for the Advanced Encryption Standard.* URL: http://csrc.nist.gov/encryption/aes/aes_home.htm. Published in the Federal Register, September 1997.

[2] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *Simple Key-Management for Internet Protocols (SKIP).* Sun Microsystems, Inc, December 1995. Work in Progress.

[3] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *Certificate Discovery Protocol.* Sun Microsystems, Inc, December 1995. Work in Progress.

[4] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *Encoding of an Unsigned Diffie-Hellman Public Value.* Sun Microsystems, Inc, December 1995. Work in Progress.

[5] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *SKIP Algorithm Discovery Protocol.* Sun Microsystems, Inc, December 1995. Work in Progress.

[6] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *SKIP Extensions for IP Multicast.* Sun Microsystems, Inc, December 1995. Work in Progress.

[7] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *X.509 Encoding of Diffie-Hellman Public Value.* Sun Microsystems, Inc, December 1995. Work in Progress.

[8] R. Atkinson. *Security Architecture for the Internet Protocol.* RFC 1825. NRL, August 1995.

[9] R. Atkinson. *IP Authentication Header.* RFC 1826, NRL, August 1995.

[10] R. Atkinson. *IP Encapsulating Security Payload.* RFC 1827,NRL, August 1995.

[11] D. Atkins, W. Stallings, P. Zimmerman. *PGP Message Exchange Formats.* Work in progress.

[12] B. Boer and A. Bosselaers. *An attack on the last two rounds of MD4.* Advances in cryptology - EUROCRYPT 93 Proceedings, Springer-Verlag, 1994.

[13] R. Braden, D. Clark, S. Crocker, and C. Huitema. *Report of IAB Workshop on Security in the Internet Architecture.* RFC 1636, USC/Information Sciences Institute, MIT, Trusted Information Systems, INRIA, June 1994.

[14] Steven M. Bellovin. *Security Problems in the TCP/IP Protocol Suite.* ACM Computer Communications Review, Vol. 19, No. 2, March 1989.

[15] Steven M. Bellovin. *Presentation at IP Security Working Group Meeting.* Proceedings of the 32nd Internet Engineering Task Force, March 1995, Internet Engineering Task Force, Danvers, MA.

[16] Steven M. Bellovin. *Using the Domain Name System for System Break-ins.* Proceedings of the Fifth Usenix UNIX Security Symposium, Salt Lake City, UT. AT&T Bell Laboratories, 1995.

[17] A. Ballardie, P. Francis and J. Crocroft. *Core Based Trees: An Architecture for Scalable Inter-Domain Multicast Routing.* Proceedings of ACM SIGCOMM 93, ACM Computer Communications Review, Volume. 23, Number 4, October 1993, pp. 85-95.

[18] Steven M. Bellovin and Michael Merritt. *Limitations of the Kerberos Authentication System.* Proceedings of USENIX Conference. Dallas, TX, Winter 1991.

[19] Eli Biham and Adi Shamir. *Differential Cryptoanalysis of DES-Like Cryptosystems.* Proceedings of *Advances in Cryptology* - CRYPTO, Springer-Verlag, 1990, pp. 2-21.

[20] Eli Biham and Adi Shamir. *Differential Cryptoanalysis of Feal and N-Hash.* Proceedings of *Advances in Cryptology* - EUROCRYPT, Springer-Verlag, 1991, pp. 1-16.

[21] Eli Biham and Adi Shamir. *Differential Cryptoanalysis of Snefru, Khafre, REDOC-II, LOKI and LUCIFER.* Proceedings of *Advances in Cryptology* - CRYPTO, 1992, pp. 156-171.

[22] Eli Biham and Adi Shamir. *Differential Cryptoanalysis of the Full 16-Round DES.* Proceedings of *Advances in Cryptology* - CRYPTO, 1993, pp. 156-171.

[23] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Willy Hacker*, Addison-Wesley, Reading, MA, 1994.

[24] CCITT Recommendation X.500. *The Directory.*

[25] CCITT Recommendation X.500. *The Directory - Authentication Framework.*

[26] *Certicom Elliptic Curve Cryptosystem Tutorials and Whitepapers.* URL: http://www.certicom.com/.

[27] *Computer Emergency Response Team.* CERT FTP Archive. URL: ftp://ftp.cert.org/pub/cert_advisories/.

[28] S. Crocker, D. Eastlake and J. Schiller. *Randomness Recommendations for Security.* RFC 1750, December 1994.

[29] Douglas E. Comer. *Internetworking with TCP/IP: Principles, Protocols and Architecture, Volume I.* Prentice-Hall, Englewood Cliffs, New Jersey, Third Edition, 1995.

[30] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls.* O' Reilly & Associates, Inc. 1995.

[31] B. den Boer and A. Bosselaers. *Collisions for the compression function of MD5.* In Advances in Cryptology - Eurocrypt '93, pages 293-304, Springer-Verlag, 1994.

[32] US Defense Intelligence Agency. *Compartmented Mode Workstation Specification*, Technical Report DDS-2600-6243-87.

[33] US National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria.* DoD 5200.28-STD, US Department of Defense, Ft. Meade, MD., December 1985.

[34] US National Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria.* NCSC-TG-005, Version 1, US Department of Defense, Ft. Meade, MD., 31 July 1987.

[35] Whitfield Diffie and Martin E. Hellman. *New Directions in Cryptography.* IEEE Transactions on Information Theory, Vol. IT-22, No. 6, November 1976, pp. 644-654.

[36] Donald E. Eastlake III and Charles W. Kaufman. *Domain Name System Security Extensions.* RFC 2065, January 1997.

[37] T. ElGamal. *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* Proceedings of *Advances in Cryptology - CRYPTO*, Springer-Verlag, 1985, pp. 10-18.

[38] Simson Garfinkel. *PGP: Pretty Good Privacy.* O' Reilly and Associates, Inc. March 1995.

[39] Galvin J., and K. McCloghrie. *Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1446, Trusted Information Systems, Hughes LAN Systems, April 1993.

[40] Simson Garfinkel and Gene Spafford. *Practical Unix Security.* O' Reilly & Associates, Inc. 1991.

[41] Stephen E. Hansen and Todd Atkins. *Automated System Monitoring and Notification with Swatch.* LISA, Monterey, CA, November 1993.

[42] N. Haller and R. Atkinson. *On Internet Authentication.* RFC 1704, Bell Communications Research, NRL, October 1994.

[43] R. Hinden and S. Deering. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 1883, December 1995.

[44] Craig A. Huegenchuegen. *The Latest in Denial of Service Attacks: Smurf.* July 1998.

[45] John Ioannidis and Matt Blaze. *Architecture and Implementation of Network-layer Security Under Unix.* Proceedings of USENIX Security Symposium, Santa Clara, CA, October 1993.

[46] John Ioannidis, Matt Blaze and Phil Karn. *swIPe: Network-Layer Security for IP.* Presentation at the Spring 1993 IETF Meeting, Columbus, Ohio.

[47] Aleksandar Jurisic and Alfred J. Menezes. *Elliptic Curves and Cryptography.* Dr. Dobb's Journal, April 1997.

[48] Burt Kaliski. *The MD2 Message-Digest Algorithm.* RFC 1319, April 1992.

[49] S. Kent. *US DoD Security Options for the Internet Protocol.* RFC 1108, BBN Communications, November 1991.

[50] S. Kent. *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* RFC 1422, BBN Communications, February 1993.

[51] Christopher William Klaus. *Stealth Scanning - bypassing Firewalls and SATAN Detectors.* Internet Security Systems, Inc, 1995.

[52] J. Kohl and B. Neuman. *The Kerberos Network Authentication Service (V5).* RFC 1510, Digital Equipment Corporation, USC/Information Sciences Institute, September 1993.

[53] P. Karn, P. Metzger and W. Simpson. *The ESP DES-CBC Transform.* RFC 1829, Qualcomm, Inc., Piermont, Daydreamer, August 1995.

[54] P. Karn and W. A. Simpson. *The Photuris Session Key Management Protocol.* Qualcomm, DayDreamer, June 1996. Work in progress.

[55] Brian A. LaMacchia and Andrew M. Odlyzko. *Computation of Discrete Logarithms in Prime Fields.* Designs, Codes, and Cryptography, 1991.

[56] Mitsuru Matsui. *Linear Cryptoanalysis Method for DES Cipher.* Proceedings of *Advances in Cryptology - EUROCYPT,* Springer-Verlag, 1994, pp. 386-397, 1994.

[57] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[58] J. Mogul and S. Deering. *Path MTU Discovery.* RFC 1191, DECWRL, Stanford University, November 1990.

[59] Ralph Merkle and Martin Hellman. *Hiding Information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory,* 24(5):525-530, September 1978.

[60] Jeffrey C. Mogul. *Simple and Flexible Datagram Access Control for UNIX based gateway*. In Proceedings of Usenix UNIX Conference, Baltimore, Maryland, summer 1989.

[61] P. Metzger and W. Simpson. *IP Authentication with Keyed MD5*. RFC 1828, Piermont, Daydreamer, August 1995.

[62] NIST (National Institute of Standards and Technology). *Secure Hash Standard (SHS)*. USA Federal Information Processing Standards Publications 180, May 1993.

[63] NIST (National Institute of Standards and Technology). *Digital Signature Standard (DSS)*. USA Federal Information Processing Standards Publications 186, May 1994.

[64] R. M. Needham and M. D. Schroeder. *Using Encryption for Authentication in Large Networks of Computers*. Communications of the ACM, Vol. 21, No. 12, December 1978, pp. 993-999.

[65] R. M. Needham and M.D. Schroeder. *Authentication Revisited*. ACM Operating Systems Review, Vol. 21, No. 1., 1981.

[66] Keesje Duarte Pouw and Paulo Lício de Geus. *Uma Análise das Vulnerabilidades dos* Firewalls. Proceeding of WAIS´96 (Workshop sobre Administração e Integração de Sistemas), Fortaleza, May, 1996.

[67] *PKCS #7 (Public Key Cryptographic Standards): Cryptographic Message Syntax Version 1.5*. RSA Laboratories, November 1993.

[68] *PKCS #10 (Public Key Cryptographic Standards): Certification Request Version 1.5*. RSA Laboratories, November 1993.

[69] J. Postel. *Internet Protocol*. RFC 791, September 1981.

[70] Routo Terada and P. G. Pinheiro. *How to Strengthen FEAL against Differential Cryptoanalysis*. Proceedings of the 1995 *Japan-Korea Workshop in Information Security & Cryptography*. Inuyama, Japan, pp. 153-162.

[71] Darren Reed. *Darren Reed home page*. ULR: http:\\coombs.anu.edu.au/~avalon/ip-filter.html. Version 3.0.2. February 1996.

[72] Sun Microsystems. *RPC: Remote Procedure Call protocol specification: Version 2*. RFC 1057, June 1988.

[73] Dareen Reed, Tom Fitzgerald and Paul Traina. *RFC 1858: Security Considerations - IP Fragment Filtering*. October, 1995.

[74] Ronald Rivest. *The MD4 Message Digest Algorithm*. RFC 1320, April 1992.

[75] Ronald Rivest. *The MD5 Message Digest Algorithm*. RFC 1321, April 1992.

[76] M. J. Robshaw. *MD2, MD4, MD5, SHA and other Hash Functions.* Technical Report TR-101, RSA Lab., 1994.

[77] Ronald Rivest, Adi Shamir and Leonard Adleman. *A method of Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120-126, February 1978.

[78] Claus Schnorr. *Efficient Signature Generation for Smart Cards.* Proceedings of *Advances in Cryptology* - CRYPTO, Springer-Verlag, 1990, pp. 239-252.

[79] Bruce Schneier. *Applied Cryptography.* John Wiley and Sons, New York, NY, 1996.

[80] *SDNS Secure Data Network System. Security Protocol 3, SP3, Document SDN.301, Revision 1.5*, 15 May 1989, published in NIST Publication NIST-IR-90-4250, February 1990.

[81] Richard Schroeppel, Hilarie Orman, Sean O'Malley and Oliver Spatscheck. *Fast Key Exchange with Elliptic Curve System.*

[82] J. Reynolds and J. Postel. *Assigned Numbers.* STD2, RFC 1700, USC/Information Sciences Institute, October 1994.

[83] Richard Stevens. *TCP/IP Illustrated.* Addison Wesley, 1994.

[84] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[85] Sun Microsystems. *Solaris 2.5 System Administrator Answerbook.* Mountain View CA, 1995.

[86] Wietse Venema. *TCP WRAPPER: Network Monitoring, Access Control and Booby Traps.* Proceedings of the Third Usenix UNIX Security Symposium, Baltimore, Maryland, September 1992.

[87] Paul Vixie. *DNS and BIND security issues.* In Proceedings of the Fifth Usenix UNIX Security Symposium, Salt Lake City, UT, 1995.

[88] V.L. Voydock and S. T. Kent. *Security Mechanisms in High-level Networks.* ACM Computing Surveys, Vol. 15, No. 2, June 1983.

[89] P. van Oorschot and M. Wiener. *Parallel collision search with application to hash functions and discrete logarithms.* In Proceedings of 2nd ACM Conference on Computer and Communication Security, 1994.

[90] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala. *RSVP: A New Resource Reservation Protocol.* IEEE Network magazine, September 1993.

[91] Marcus J. Ranum. *Firewalls FAQ—Frequently Asked Questions.* 1995.

[92] Frank Willoughby. *Internet Firewall Vulnerabilities.* Fortified Networks Inc. November 1995

[93] Alfred J. Menezes, Paul C. Van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications, October 1996.

[94] Cláudio L. Lucchesi. *Introdução à Criptografia Computacional*. Papirus Editora, Campinas SP, 1986.

[95] Steven M. Bellovin. *Problem Areas for the IP Security Protocols*. Proceedings of the Sixth Usenix UNIX Security Symposium, San Jose, CA. July, 1996

[96] J. Linn. *Generic Security Service Application Program Interface, Version 2*. RFC 2078. OpenVision, January 1997.

[97] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. *The Private Communication Technology Protocol, Version 1.00*. Microsoft Corporation, Redmond, WA. October, 1995.

[98] Alan O. Freier, Philip Karlton and Paul C. Kocher. *The SSL Protocol Version 3.0*. Netscape Corp. November, 1996.

[99] J. Linn. *Generic Security Service Application Program Interface*. RFC 1508. OpenVision, September 1993.

[100] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. RFC 1421. February, 1993.

[101] Simson Garfinkel and Gene Spafford. *Practical Unix Security & Internet Security, 2nd Edition*. O' Reilly & Associates, Inc., 1996.

[102] P. McMahon. *GSS-API Authentication Method for SOCKS Version 5*. RFC1921, June 1996.

[103] C. Adams. *The Simple Public-Key GSS-API Mechanism (SPKM)*. RFC2025, October 1996.

[104] P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*. April 1997.

[105] *SET Secure Electronic Transaction Specification. Book 1: Business Description Version 1.0*. Visa and MasterCard. May 31, 1997.

[106] T. Ylonen. *SSH—Secure Login Connections over the Internet*. SSH Communications Security Ltd, Finland. July, 1996.

[107] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne and S. Lehtinen. *SSH Protocol Architecture*. August 6, 1998. Work in Progress.

[108] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne and S. Lehtinen. *SSH Transport Layer Protocol*. August 6, 1998. Work in Progress.

[109] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne and S. Lehtinen. *SSH Authentication Protocol.* August 6, 1998. Work in Progress.

[110] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne and S. Lehtinen. *SSH Connection Protocol.* August 6, 1998. Work in Progress.

[111] P. Deutsch and J-L. Gailly. *ZLIB Compressed Data Format Specification version 3.3.* RFC 1950. May 1996.

[112] Paul Vixie. *Name Server Operations Guide for BIND Release 4.9.4.* Internet Software Consortium, 1996.

Apêndice A

Algoritmos de criptografia

Algoritmo	Descrição	Criptoanálise	Patente
LUCIFER IBM, 1970.	Precursor do DES. Introduziu o conceito de **Rede de Feistel**. Chave e Blocos de 128 bits. 16 iterações.	Sujeito à Criptoanálise Diferencial[BS92].	Patente vencida.
FEAL NTT Japão, 1987.	Objetivo do projeto: função de iteração mais forte para obter um nível de segurança equivalente ao DES, porém com um número menor de iterações. Chave e bloco de 64bits (versão original). FEAL-NX apresenta chave de 128 bits.	Diversas versões do algoritmo foram criptoanalisadas com sucesso: FEAL-8 (8 iterações), FEAL-16, FEAL-NX[BS91]. Uma variante do algoritmo é resistente à criptoanálise diferencial: FEAL-N(X)S [TP95].	licença: NTT, 1-6 Uchsaiwai-cho, 1-chome, Chiyada-ku, 100 Japan.
DES 3DES	A variante 3DES é implementada da seguinte forma: $C = E_{K1}[D_{K2}[E_{K1}[P]]]$.	O TRIPLE-DES tem chave equivalente a 112 bits[KM92]. O DES é resistente à criptoanálise diferencial, no entanto, é mais susceptível à criptoanálise linear[BS93].	Não se Aplica. Padrão ANSI.
RC2 RSADSI R. Rivest, 1994.	Chave de tamanho variável e bloco de 64 bits. Cifrador não iterativo diferentemente do DES e similares. Não possui S-Boxes. Os detalhes do projeto são segredos comerciais.	Analisado apenas pela empresa que o projetou: RSADSI. Usado em vários produtos comerciais (implementação SSL de browsers).	Não se aplica. Protegido como segredo comercial. Licenças: RSADSI.

Tabela A.1: Algoritmos simétricos.

Algoritmo	Descrição	Criptoanálise	Patente
RC4 RSADSI R. Rivest, 1987.	*Stream cipher* com tamanho de chave variável. Opera em modo OFB. Os detalhes do algoritmo eram segredo comercial até 1994, quando seu código fonte foi publicado na Internet.	Nenhum resultado de criptoanálise linear ou diferencial é conhecido contra o RC4. Também é muito utilizado comercialmente.	Após a publicação de seu código fonte tornou-se de domínio público. No entanto, o nome RC4 é ainda marca registrada da RSADSI.
RC5 RSADSI R. Rivest, 1995.	Aceita 3 parâmetros variáveis: tamanho da chave, bloco e número de iterações. É composto de 3 operações básicas: OU-Exclusivo, adição e rotação.	Resistente à criptoanálise diferencial quando o número de iterações é superior a 15 [Sch96, p. 346].	Patente da RSADSI.
IDEA Xuejia Lai e James Massey, Suíça 1992.	Algoritmo iterativo. Estrutura em rede de Feistel. Chave de 128 bits e bloco de 64 bits. Operações básicas: OU-Exclusivo, Adição módulo 2^{16} e Multiplicação módulo $2^{16} + 1$. Baseado em forte embasamento teórico. Duas vezes mais rápido que o DES.	Nenhum ataque bem sucedido para o número especificado de iterações (8). Considerado um dos melhores algoritmos simétricos de bloco disponíveis ao público.	Liberado para uso não comercial. licença: Ascom, Gewerbepark, CH-5506, Mägenwil, Switzerland; idea@ascom.ch.
BLOWFISH Schneier 1994.	Estrutura também baseada em rede de Feistel. Chave variável de até 448 bits, bloco de 64 bits e 16 iterações. Operações básicas: permutações dependentes da chave e substituições (S-Boxes) dependentes do texto claro.	Nenhum resultado de criptoanálise conhecido para o número original de iterações (16).	Não patenteado.
AES - Advanced Encryption Standard. NIST[a]: a definir.	Proposta de definição de padrão para substituir o DES. Algoritmo de bloco de 128 bits, e chave de tamanho variável: 128, 192 ou 256 bits. Concurso público para a escolha do algoritmo definitivo.		

Tabela A.1: Algoritmos simétricos.

a. National Institute of Standards and Technology (EUA).

Algoritmo	Descrição	Criptoanálise	Patente
KNAPSACK Hellman e Merkle, 1978.	Baseado no problema da Mochila. Problema NP-completo[MH78].	Todas as variantes deste algoritmo são inseguras.	Patentes nos EUA[a] e Europa. Patente americana vence em 1997.
MCELICE Robert McEliece, 1978.	Baseado na teoria de codificação algébrica. De duas a três vezes mais rápido que o RSA. Chave pública muito grande: 2^{19} bits de comprimento. O texto cifrado é duas vezes maior que o texto claro original.	Princípio semelhante ao algoritmo KNAPSACK. Contudo, nenhum ataque foi bem sucedido contra este algoritmo. Pouco divulgado.	Não patenteado.
ELGAMAL ElGamal, 1985.	Pode ser usado tanto para gerar assinaturas digitais como para cifrar dados. Baseado no problema do logaritmo discreto [ElG85]. Mesmo princípio do primeiro algoritmo assimétrico proposto por Diffie-Hellman[DH76].	Intratabilidade do problema do logaritmo discreto em um corpo finito.	Não patenteado.
DSA NIST, 1991 (proposto).	Algoritmo para gerar assinaturas digitais proposto pelo governo dos EUA[NIST94]. Esquema de assinaturas baseado no problema do logaritmo discreto. Semelhante ao algoritmo de ELGAMAL e SCHNORR[Sch90].	Intratabilidade do problema do logaritmo discreto em um corpo finito.	Patenteado nos EUA; licença livre. Pendência em relação às patentes: Diffie-Hellman, Merkle-Hellman e Schnorr.
ECC - Elliptic Curve Cryptosystems.	Sistemas de criptografia assimétricos definidos sobre corpos compostos de pontos de uma curva elíptica.	Intratabilidade do problema do logaritmo discreto em grupos aritméticos definidos sobre os pontos de uma curva elíptica. Oferecem o mesmo nível de segurança que os sistemas baseados em corpo de inteiros, mas com tamanho de chave menor. Um ECC de 160 bits equivale a um sistema RSA de 1024 bits.	Algumas patentes para cálculos específicos e otimizações.

Tabela A.2: Algoritmos assimétricos.

a. Esta patente supostamente cobria todos os sistemas de chave pública nos EUA.

Apêndice B

Desenvolvimento de aplicações seguras usando HTML/HTTPS

B.1 Resumo

Este artigo apresenta diretrizes para o desenvolvimento de aplicações seguras em ambiente HTML/HTTPS. A separação em camadas bem definidas, com funções de segurança não sobrepostas, juntamente com uma série de cuidados inerentes ao serviço sendo oferecido, permitem atingir o desejado grau de segurança para aplicações com transferências de valores, destacando-se Internet *banking* e comércio eletrônico em geral.

As camadas propostas são analisadas com relação às suas funções no processo e aos aspectos de segurança envolvidos, obtendo-se como resultado um modelo para a construção de aplicações consistentes, de alto nível de segurança, na Web. Conclui-se que o problema de segurança não pode ser resolvido apenas tratando-se individualmente ou coletivamente parte dos elementos envolvidos, pois muitas vezes até a própria aplicação acaba se constituindo no elo mais fraco, por descuido no projeto global.

B.2 Introdução

O advento do protocolo HTTPS—protocolo HTTP (*Hyper Transfer Transmission Protocol*) [1] sobre SSL (*Secure Socket Layer*) [2]—representou um divisor de águas no desenvolvimento de aplicações que requerem premissas básicas de segurança, notoriamente os conceitos de privacidade, integridade e autenticação. Desde a sua proposição inicial pela Netscape em meados de

1996, o protocolo SSL tornou-se um padrão de fato para a construção de canais de segurança de maneira transparente para as aplicações, em especial àquelas desenvolvidas sobre o paradigma HTTP, cujo modelo de aplicações será analisado, bem como suas vulnerabilidades nativas.

A construção de aplicações críticas sobre este paradigma, tais como banco eletrônico e comércio eletrônico em geral, envolve questões mais sutis que o simples uso do protocolo SSL e mecanismos nativos disponíveis nos *browsers* comerciais. A integração com os sistemas legados, autenticação de clientes e formalização de não-repúdio são premissas importantes, porém não nativas nos sistemas atuais (*browsers* e servidores comercias), e que serão analisadas neste trabalho.

Vale ressaltar que estas tecnologias são disponibilizadas nos clientes Internet através de aplicações de cunho genérico, popularmente denominadas *browsers*. Apesar dos *browsers* tentarem passar o conceito de cliente universal através da padronização da linguagem de interface HTML (*HyperText Markup Language*), na prática detalhes da tecnologia de cada *browser* e questões de segurança em geral devem ser considerados na construção de aplicações Web específicas. Os *browsers* de mercado mais populares (90% do mercado)—Netscape Communicator [3] e Microsoft Internet Explorer [4]—serão especificamente abordados neste documento.

B.3 Cenário

O cenário típico de aplicações seguras na Internet envolve o protocolo HTTP sobre SSL. Tipicamente a interface do usuário é codificada em HTML, e lógicas simples no cliente são implementadas através de linguagens de *script* que possuem interface nativa com HTML nos *browsers* comerciais, tais como JavaScript [5] e VBScript [6]. Lógicas de aplicação mais sofisticadas somente podem ser obtidas através de mecanismos dependentes do *browser* utilizado. Notadamente, a tecnologia de Plugin para o *browser* Communicator e ActiveX para o *browser* Internet Explorer. A Figura B.1 ilustra a relação entre a camada de protocolos da arquitetura TCP e a arquitetura genérica de aplicação descrita nas seções seguintes.:

B.4 Protocolo SSL

O protocolo SSL provê uma abstração de canal seguro de comunicação, ou *stream* seguro. Os princípios de integridade, privacidade e autenticação de servidor são oferecidos à interface HTML

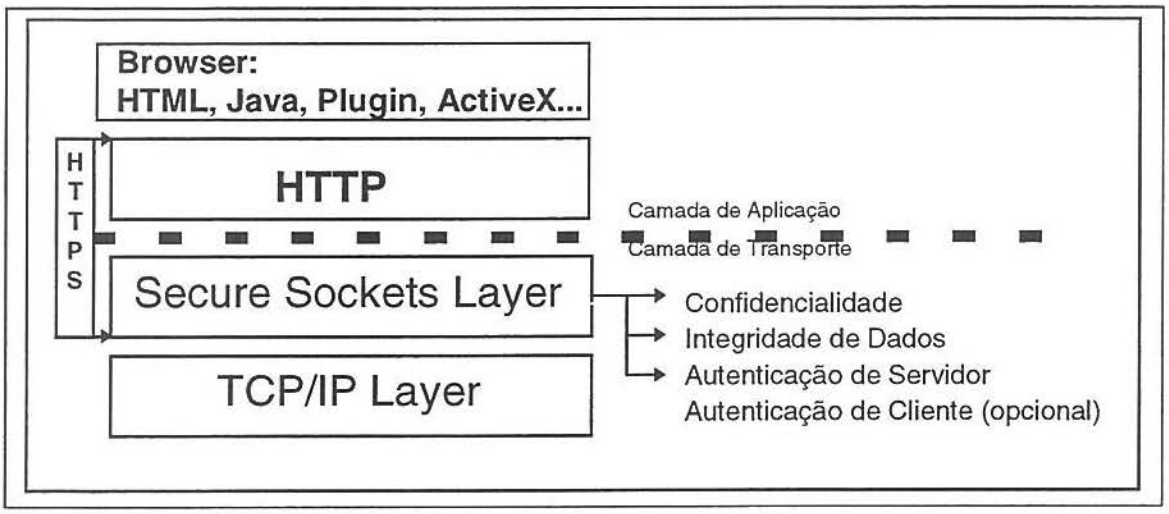

Figura B.1: Relação entre as arquiteturas TCP/IP e a genérica de aplicação.

de maneira transparente. Para criar um canal seguro com um servidor qualquer, a URL (*Unified Resource Locator*) deve referenciar seu nome precedido do prefixo HTTPS, por exemplo, `https://www.unicamp.br`.

A autenticação de clientes, no entanto, requer a geração de certificados digitais de maneira específica para cada *browser* de mercado. De maneira geral, a autenticação de clientes baseada em certificados SSL é muito pouco difundida, mesmo para aplicações críticas que envolvam transferência de valores. A falta de padronização na interface de uso e geração de certificados, além da complexidade de implementação e uso em comparação à utilização de senhas convencionais, tem inibido o uso mais extensivo desta técnica de autenticação.

Por outro lado, o princípio de não repúdio não tem como ser implementado de maneira transparente pelo protocolo SSL e, como veremos adiante, requer um alto grau de integração e infra-estrutura de segurança sofisticada por parte da aplicação em si.

Ainda assim, a principal deficiência da arquitetura de segurança HTTPS está na fraca associação, por parte dos *browsers* comerciais, entre uma conexão segura e o servidor seguro propriamente dito. Tal vulnerabilidade é denominada *link spoofing*, e será descrita a seguir.

B.5 *Link spoofing*

Na arquitetura SSL a autenticação de servidores é efetuada através de certificados públicos emitidos por entidades certificadoras (*Certificate Authority*—CA), cujos próprios certificados encontram-se estaticamente armazenados nas aplicações clientes (*browsers*). Este esquema oferece uma maneira segura de associar uma URL, por exemplo `https://wwws.companiax.com`, ao seu respectivo servidor seguro. Isto se o usuário explicitamente abrir a URL em questão. Neste caso, um canal seguro de comunicação é estabelecido entre o servidor citado na URL e o usuário que a referenciou.

No entanto, uma prática extremamente comum na Internet é "navegar" de referência em referência (*link*) até uma página transacional segura obtida através do protocolo HTTPS. Como a página que referencia o link seguro é transferida através do protocolo HTTP, esta está sujeita às vulnerabilidades do protocolo TCP/IP. Ou seja, a referência ao link seguro original pode ser alterada em trânsito ou através de técnicas de personificação de máquinas descritas em [7]. Assim, um usuário, ao acessar a referência adulterada (possivelmente também uma referência HTTPS), pode ser conduzido a uma outra página, provavelmente em outro servidor e possivelmente com o mesmo aspecto gráfico da referência original.

Se a referência adulterada for também uma referência HTTPS, então o indicador de conexão segura nos *browsers* comerciais mostrar-se-á como o esperado. Neste caso, a única proteção oferecida ao usuário é a visualização dos atributos de segurança (certificado SSL do servidor) da URL em questão, o que constitui prática pouco comum entre a maioria dos usuários.

B.6 Protocolo HTTP-HTML

O HTTP é um protocolo simples orientado a mensagens. O cliente estabelece uma conexão TCP com um servidor, envia um comando, e recebe os dados do servidor. A conexão TCP é finalizada pelo servidor assim que os dados são transmitidos[1]. A mensagem retornada pelo servidor é normalmente um documento HTML com referências (*links*) para outros documentos HTML, imagens, arquivos PostScript, arquivos texto etc.

1. A versão 1.1 do protocolo HTTP cria o conceito de conexões persistentes, onde várias mensagens são enviadas em uma única conexão TCP [1].

Os principais comandos HTTP são os métodos GET e POST. O método GET retorna qualquer dado associado à URL em questão. O método POST, por sua vez, é usado para enviar correio eletrônico (e-mail), e principalmente formulários eletrônicos preenchidos interativamente pelos usuários. Este é o único comando que envia parâmetros de requisição.

Os formulários eletrônicos construídos através das linguagens HTML e Java, e de componentes ativos formatam os parâmetros em uma sintaxe conhecida informalmente como *label*-valor. Nesta estrutura cada parâmetro recebe um identificador ao qual o valor, após ser codificado[1], será associado. O caracter "=" associa um label a um determinado valor e o caracter "&" separa os pares *label*-valor. O objetivo deste padrão é disponibilizar um "protocolo universal" de troca de dados entre aplicações cliente-servidor sem estrutura pré-definida.

A interface entre o servidor Web e as aplicações residentes na máquina servidora é feita através de componentes conhecidos genericamente como CGIs (*Common Gateway Interface*). Os cabeçalhos HTTP, comando e parâmetro (comando POST) são normalmente obtidos através de variáveis de ambiente e repassados às aplicações que interpretam e tratam a estrutura *label*-valor. Os CGIs podem ser fisicamente executáveis ou extensões dos servidores Web na forma de bibliotecas compartilhadas (Microsoft ISAPI e Netscape NS-API), dentre outras tecnologias que fogem ao escopo deste trabalho.

B.7 Componentes ativos

Entende-se por componentes ativos todo e qualquer código atualizado dinamicamente nos clientes Internet (*browsers*) que tenha acesso a recursos de sistema, tais como os recursos de entrada e saída (I/O) e arquivos. Claramente, tais conteúdos, normalmente embutidos no contexto de páginas HTML, podem introduzir vírus, cavalos de Tróia e outros elementos nocivos em seus sistemas hospedeiros. Os componentes ativos mais difundidos são os controles ActiveX (Microsoft) [8] e Plugins (Netscape) [9].

Normalmente a atualização/instalação de componentes ativos nos *browsers* somente é permitida se os mesmos forem eletronicamente assinados com a chave pública de alguma entidade confiável (*Certificate Authority*—CA). No entanto, a exibição do certificado da entidade emissora do conteúdo ativo somente prova sua autenticidade, mas não diz a nada a respeito do conteúdo

1. Por *default*, o padrão normalmente utilizado pelos *browsers* e aplicações em geral é URL *encoding* [10].

em si. Assim a melhor proteção é simplesmente desabilitar a execução de conteúdos ativos nos *browsers*, ou filtrá-los nos *firewalls* quando estes estiverem disponíveis.

B.8 Arquitetura de aplicação Web segura

Infelizmente o processo de construção de aplicações Web seguras é menos determinístico do que aparenta em uma primeira análise. Entretanto, algumas diretrizes básicas podem tornar este trabalho mais palpável no contexto de aplicações Web, isto é, no ambiente HTTPS. A arquitetura, projeto e programação em um ambiente qualquer influi diretamente na qualidade e segurança do sistema desenvolvido. De uma maneira geral, o ambiente Web é constituído de uma interface Web, um servidor de aplicação e os sistemas legados, conforme ilustrado na Figura B.2.

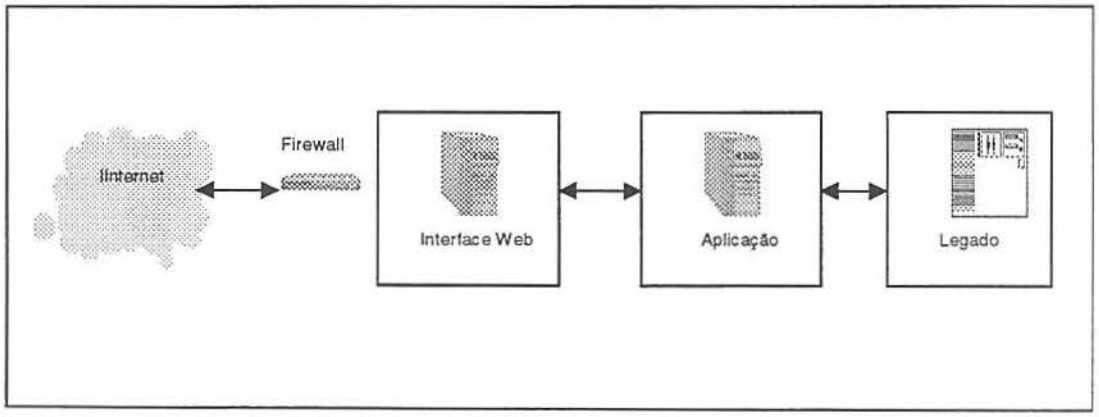

Figura B.2: Arquitetura de aplicação Web segura em ambiente típico.

B.8.1 Interface Web

Refere-se a todos os aplicativos e serviços acessíveis via Internet, tais como o servidor Web propriamente dito, servidor FTP, correio eletrônico etc. Como estes elementos têm contato direto com o mundo externo e com a rede interna, eles devem estar instalados em máquinas fortificadas (*bastion hosts*) e isolados em uma **zona desmilitarizada** [10].

Desta forma, apenas os serviços disponibilizados externamente terão regra de acesso definida no *firewall* para as máquinas na zona desmilitarizada, e estas por sua vez acessarão apenas os serviços de aplicação definidos na rede interna. Assim, mesmo que a interface Web seja comprometida por falhas de *software* (*bug* no servidor Web, por exemplo), os sistemas legados e rede interna não serão comprometidos. Um *firewall* que disponha de uma sub-rede exclusiva para os

bastion hosts, com controle de acesso tanto para a Internet quanto para a rede interna, é tecnicamente denominado de *screened subnet* [10].

Esta última premissa de segurança somente será verdadeira se uma outra premissa de desenvolvimento de sistemas for respeitada: se a **camada de apresentação**, que corresponde na prática à construção de páginas HTML dinâmicas, estiver perfeitamente isolada do contexto de aplicação na interface Web. Isto porque os mecanismos de autenticação e de acesso ao banco de dados e sistemas legados não devem ser acessíveis pela camada de apresentação. Além disso, como o processo de alteração de "visual" é muito dinâmico e freqüente, erros neste processo poderiam ocasionar falhas de segurança, caso alguma lógica de aplicação estivesse acoplada ao mesmo.

B.8.2 Servidor de aplicação.

O serviço de aplicação deve ser responsável por disponibilizar e atualizar as informações requeridas pela Web, assegurando integridade e segurança às mesmas. Novamente, para assegurar o cumprimento de suas responsabilidades este serviço deve, preferencialmente, ser estruturado em camadas.

De fato, o serviço de aplicação pode ser entendido como um **servidor transacional** que deverá efetuar três verificações básicas: identificar o componente que implementa a função requerida (por exemplo, uma consulta qualquer), verificar as permissões de acesso e instanciar/executar a função requerida. Além disso, serviços básicos podem ser disponibilizados aos componentes específicos e à camada de controle de acesso. Tipicamente temos os serviços de gerenciamento de sessão, autenticação e camada de acesso ao banco de dados. A Figura B.3 ilustra esta arquitetura:

Note que nesta arquitetura as funções do sistema somente poderão ser executadas através do servidor transacional, que automaticamente cuidará de efetuar as devidas verificações de segurança e controle de acesso. Desta forma, novas funcionalidades podem ser agregadas automaticamente sem riscos de falhas de segurança e sem o custo de implementação de mecanismos de segurança específicos.

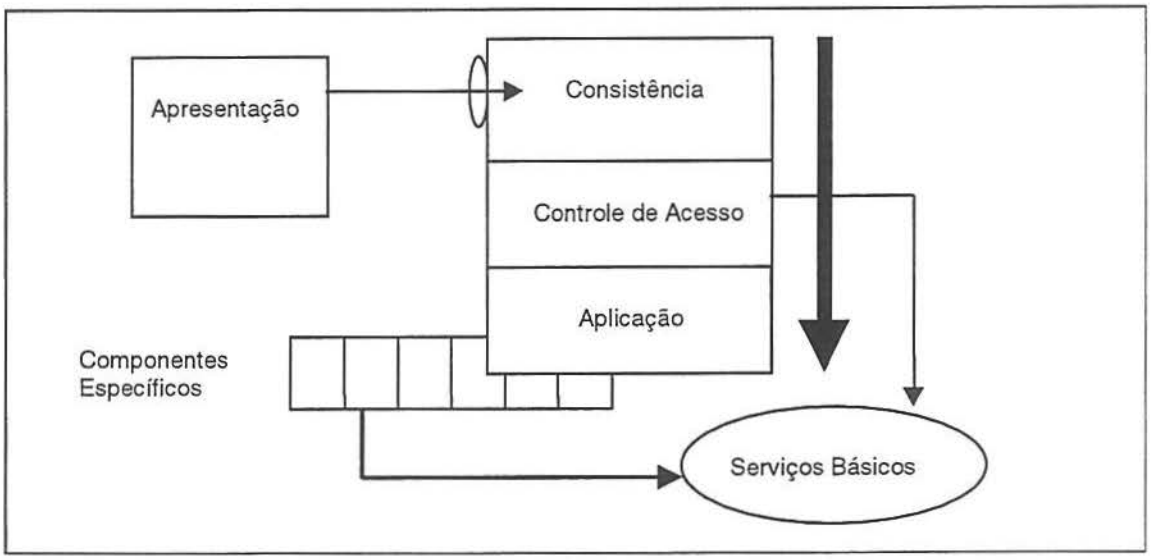

Figura B.3: Arquitetura de servidor de aplicação.

B.8.3 Controle de sessão

Esta camada cria a abstração de sessão para os protocolos não orientados à conexão (tipicamente HTTP) e comandos assíncronos. Associa o contexto de sessão (sessionID) com os parâmetros de autenticação (usuário, método de autenticação, bilhete de acesso), *timeout* e área comum de dados entre aplicações e métodos. A camada de controle de acesso utiliza-se diretamente dos parâmetros de autenticação para determinar se uma dada requisição tem permissões de execução ou não (Figura B.4).

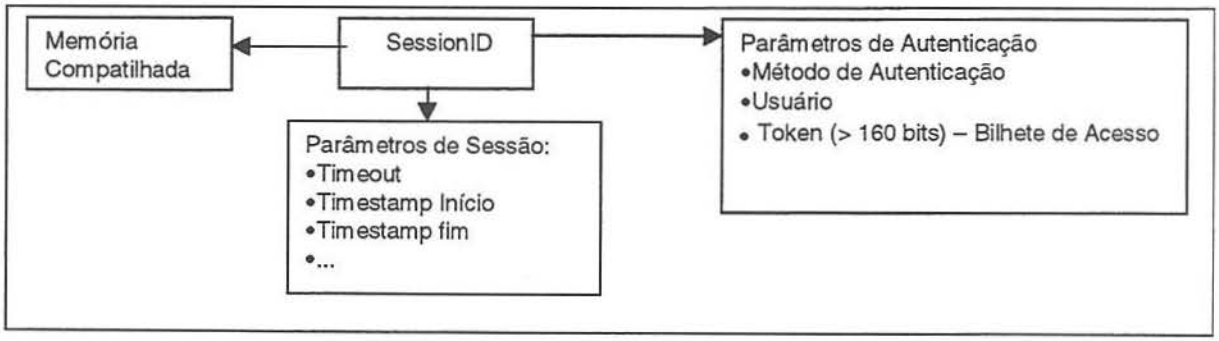

Figura B.4: Camada de controle de acesso (sessão).

B.8.4 Autenticação e não-repúdio

Devido à certa facilidade existente em personificar um servidor Web seguro (*spoofing*), técnicas de autenticação baseadas em senhas reutilizáveis tornam-se extremamente frágeis.

Vejamos, por exemplo, o seguinte cenário de ataque: em uma página institucional insegura (`http://companhiax.com.br`, por exemplo) aparece o link para a "página transacional segura", digamos,

`https://wwws.companhiax.com.br/scripts/internet2.dll?Pag=Login`.

Como a página institucional é insegura, o *link* anterior poderia ser substituído para, digamos,

`https://wwws.ciax.com.br/scripts/internet2.dll?Pag=Login`.

Obviamente, o intruso teria que registrar um domínio (no caso, `wwws.ciax.com.br`) para tal fim[1]. Ao entrar no *link* transacional falso, o usuário enviaria sua senha inadvertidamente. Após capturar o segredo em questão, o servidor atacante poderia simplesmente interromper a conexão, simulando uma falha de comunicação, ou redirecionando o usuário para o servidor correto.

Portanto, no ambiente HTTPS, aplicações críticas do ponto de vista de segurança devem implementar métodos de autenticação baseados no conceito de *"one time passwords"*. Dentre estes métodos destacam-se os baseados em mecanismo de desafio/resposta (*challenge/response*) [11] e aqueles baseados em assinaturas digitais [10]. Estes últimos, quando derivados de algoritmos assimétricos e devidamente implementados, garantem também a propriedade de não-repúdio, especialmente necessária em transações que envolvam transferência de valores.

A implementação de não-repúdio exige integração (interface) direta da aplicação com os módulos de criptografia para o cálculo/verificação da assinatura digital de mensagens específicas. Ou seja, o cliente Web (*browser*) deve ser capaz de identificar quais mensagens precisam ser assinadas e o servidor (controle de acesso) identificar quais mensagens deveriam vir assinadas e verificar tais assinaturas. Além disso, seqüenciadores aleatórios devem estar associados a cada mensagem de maneira a evitar ataques de repetição de mensagens, e todas as mensagens assinadas devem também ser armazenadas para eventual verificação.

Vale ressaltar que o protocolo SSL apresenta mecanismos nativos para autenticação de clientes baseados em certificados digitais (assinatura digital), de maneira transparente para a camada de aplicação. Por outro lado, estes mecanismos fogem do princípio de isolamento da interface

---

1. Cabe aos órgãos responsáveis na Internet o zelo com relação ao registro de nomes, principalmente os similares. No Brasil, a FAPESP passou há algum tempo a exigir CGC para o registro de nomes no domínio com.br.

Web e aplicação, uma vez que a implementações de SSL dos servidores Web validam internamente as assinaturas digitais recebidas e simplesmente repassam a identidade do cliente para as CGIs.

Ainda assim, os mecanismos de autenticação de cliente do SSL não provêem recursos nativos para a implementação de assinaturas digitais sob requisição da aplicação, e consequentemente para não-repúdio. Só recentemente a última versão do *browser* Communicator (4.5) dispôs mecanismos para assinar requisições sob demanda (*Form Signing*) [12], utilizando-se da chave privada e certificado público de clientes SSL. Até então os mecanismos para calcular a assinatura digital de requisições nos *browsers* tinham que ser desenvolvidos através de componentes ativos, que como fora abordado anteriormente, apresentam diversas vulnerabilidades para a segurança dos sistemas que os hospedam.

B.8.5 Integração com sistemas legados

As mensagens recebidas pela camada de aplicação Web seguem o formato *label*-valor descrito anteriormente. No entanto, sistemas pré-existentes que interajam com as aplicações Web podem ter estrutura de mensagens próprias para as quais os primeiros devem ser convertidos. Como regra geral, este processo deve ser feito de maneira muito criteriosa.

Como a estrutura *label*-valor pode ser facilmente editada e modificada pelos usuários, todos os *labels* recebidos, que não sejam de responsabilidade de preenchimento do usuário, devem ser cuidadosamente consistidos para cada requisição recebida. Preferencialmente, cada requisição (comando) deve saber identificar exatamente quais *labels* são esperados e quais devem ser consistidos, além de identificar *labels* repetidos e valores fora da faixa esperada.

Suponhamos o seguinte cenário: o controle de acesso verifica e valida o identificador de sessão (sessionID) e bilhete de acesso recebidos. Em seguida, a identidade do usuário (userID) dono da sessão em questão seria acrescentada à estrutura *label*-valor, antes deste ser repassado para a camada responsável por converter esta estrutura no formato de mensagem esperado pelo sistema legado. Este fluxo é representado na Figura B.5:

Neste cenário temos uma potencial vulnerabilidade que é descrita a seguir. Suponhamos que um usuário mal intencionado altere a requisição original de, por exemplo, uma aplicação de Internet *banking*, trocando a informação da conta requisitada por uma outra pertencente a outro

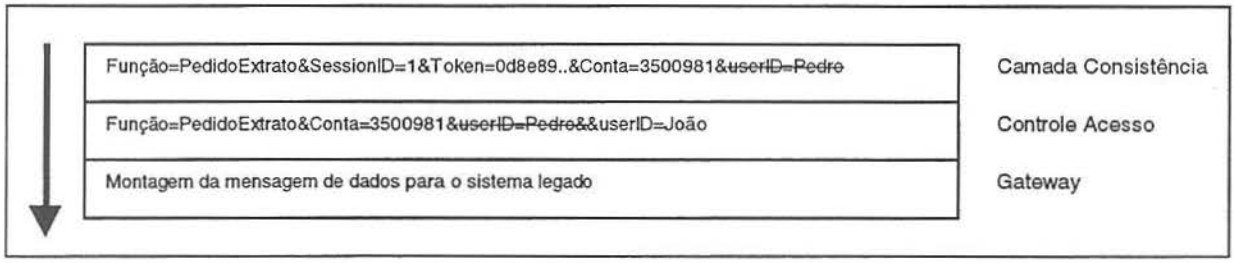

Figura B.5: Fluxo de informação em direção a sistema legado.

usuário, e acrescente ao final da requisição o identificador de usuário (userID) do proprietário da conta alterada.

Seguindo o modelo da estrutura acima, o "dono verdadeiro" da sessão é acrescentado à estrutura *label*-valor antes de ser repassada ao Gateway de acesso. Se a camada de Controle de acesso não identificar a pré-existência do *label* userID, e o Gateway não identificar a duplicidade de *labels* e procurar pelo primeiro *label* userID, a mensagem enviada ao sistema legado incluirá as informações da conta adulterada e não da do usuário que de fato tenha se autenticado. Naturalmente, esta é uma situação específica que pressupõe um conhecimento prévio do sistema a ser atacado. Contudo, isto ilustra as nuances existentes no processo de validação/autorização de requisições, bem com no tratamento da estrutura de dados e interfaces com os sistemas legados.

B.9 Considerações finais

A existência do protocolo SSL de maneira nativa nos *browsers* comerciais, e a recente liberação por parte do governo norte-americano à exportação de algoritmos simétricos para instituições financeiras[1] [13], trouxe grande impulso ao desenvolvimento de aplicações sensíveis do ponto de vista de segurança, principalmente no que tange à transferência de valores.

No entanto, a existência de um canal seguro entre clientes e servidores na Internet não é suficiente para garantir todas as premissas de segurança necessárias em aplicações do gênero (financeiras). Autenticação forte, baseada em assinaturas digitais ou mecanismos de desafio/resposta, é estritamente necessária devido à facilidade que o ambiente apresenta para personificar servidores seguros (*server spoofing*). Ainda assim, mecanismos de assinatura digital são cruciais

1. A referência aqui é obviamente aos algoritmos de criptografia forte (128 bits), em comparação com os anteriores liberados à exportação, de apenas 40 bits.

para obter a propriedade de não-repúdio. Além disso, uma infra-estrutura de sequenciamento de mensagens e armazenamento das mesmas deve ser construída com tal finalidade.

Conclui-se que uma arquitetura específica de aplicação Web, segmentação em camadas de autenticação, controle de acesso, armazenamento e análise de mensagens (normalmente sua ausência) são elementos cruciais, porém não prontamente disponíveis e pouco difundidos. De fato, de pouco vale o uso de técnicas avançadas de segurança e criptografia se não forem respeitados os princípios básicos, de qualidade, arquitetura e engenharia de *software*, com foco em segurança de sistemas, e não apenas de serviços (*firewalls*) ou canais (criptografia).

B.10 Referências

[1] W. Richard Stevens. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the Unix Domain Protocols*. Addison-Wesley, 1996.

[2] Alan O. Freier, Philip Karlton and Paul C. Kocher. *The SSL Protocol Version 3.0*. Netscape. November, 1996.

[3] http://home.netscape.com/browsers/index.html

[4] http://www.microsoft.com/windows/ie/

[5] David Flanagan. *Javascript: The Definitive Guide, 3rd Edition*. O'Reilly & Associates. 1998.

[6] Eric Smith, et al. *Inside VBScript with ActiveX*. New Riders Publishing. 1997.

[7] Keesje D. Pouw e Paulo L. de Geus. Uma Análise das Vulnerabilidades dos Firewalls. WAIS´96 (Workshop sobre Administração e Integração de Sistemas), Fortaleza. Maio, 1996.

[8] Adam Denning. *ActiveX Controls Inside Out*. Microsoft Press, 1997.

[9] Mike Morgan. *Netscape Plug-Ins Developer's Kit*. 1997.

[10] Keesje Duarte Pouw. *Segurança na Arquitetura TCP/IP: de Firewalls a Canais Seguros*. Tese de Mestrado, IC-Unicamp. Janeiro, 1999.

[11] N. Haller. *The S/KEY One-time Password System*. RFC 1760. Bellcore. February, 1995.

[12] http://developer.netscape.com/docs/manuals/security/sgntxt/index.htm

[13] Microsoft Corp. *Server Gated Cryptography*. http://www.microsoft.com/windows/ie/security/sgc.asp