UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

USANDO O ENCADEAMENTO DE TRANSAÇÕES PARA IMPLEMENTAR UM CONTROLE FIM-A-FIM DE FRAUDES EM APLICAÇÕES DISTRIBUÍDAS

ROSTAND EDSON OLIVEIRA COSTA

Campina Grande
1999

UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

ROSTAND EDSON OLIVEIRA COSTA

USANDO O ENCADEAMENTO DE TRANSAÇÕES PARA IMPLEMENTAR UM CONTROLE FIM-A-FIM DE FRAUDES EM APLICAÇÕES DISTRIBUÍDAS

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus II, como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Orientador: **Prof. Francisco Vilar Brasileiro, Ph. D**
Linha de Pesquisa: **Engenharia de Software**
Área de Concentração: **Ciência da Computação**

Campina Grande
1999

C837u Costa, Rostand Edson Oliveira
Usando o encadeamento de transacoes para implementar um controle fim-a-fim de fraudes em aplicacoes distribuidas / Rostand Edson Oliveira Costa. - Campina Grande, 1999.
99 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Seguranca Computacional 2. Deteccao de Fraudes 3. Aplicacoes Distribuidas 4. Dissertacao - Informatica I. Brasileiro, Francisco Vilar II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.056(043)

USANDO O ENCADEAMENTO DE TRANSAÇÕES PARA IMPLEMENTAR UM CONTROLE FIM-A-FIM DE FRAUDES EM APLICAÇÕES DISTRIBUÍDAS

ROSTAND EDSON OLIVEIRA COSTA

DISSERTAÇÃO APROVADA EM 06.04.1999

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador

PROF. MARCELO ALVES DE BARROS, Dr.
Examinador

PROF. JOAQUIM CELESTINO JÚNIOR, Dr.
Examinador

CAMPINA GRANDE – PB

Aos meus filhos, Giulia e Renan.

Resumo

Este trabalho apresenta um algoritmo para detecção de fraudes que pode ser utilizado em aplicações distribuídas, envolvendo transações eletrônicas ligadas ao comércio eletrônico ou não, a ser incorporado na camada de ap'licação. Baseado em um conceito simples e de fácil implementação, o encadeamento de transações permite a detecção de fraudes de origem interna ou externa, mesmo posteriormente ao processamento da transação. O uso de tal mecanismo fim-a-fim, de forma acessória e complementar aos demais mecanismos de segurança eventualmente adotadas pela aplicação, também propicia ganhos adicionais na prevenção de fraudes e no nível de tolerância a falhas do sistema. Uma aplicação exemplo é utilizada para demonstrar a aplicabilidade do algoritmo e para melhorar o entendimento dos conceitos e técnicas discutidos.

Abstract

This work presents an algorithm that can be incorporated into the application layer for detection of frauds in distributed applications that deal with electronic transactions. Based in a simple concept and easy implementation, the chaining of transactions allows the detection of frauds of both internal and external origin, even after the processing of the transaction. The use of this accessory mechanism in the end-to-end layer to complement other security mechanisms that may be adopted by the application also propitiates additional gains in the prevention of frauds and in the level of fault tolerance of the system. An example application is used to demonstrate the applicability of the algorithm and to improve the understanding of the concepts and techniques.

Agradecimentos

Ao professor Gilvandro de Oliveira Rodrigues Filho, pela contribuição direta e indireta, tanto através de sugestões e críticas quanto pela viabilização de uma importante parcela das condições necessárias para a realização deste trabalho.

Aos administradores do PARAIBAN – Banco do Estado da Paraíba e do UNIPÊ – Centro Universitário de João Pessoa pela compreensão e liberação parcial das minhas atividades em etapas importantes deste trabalho.

A Fábio Albuquerque, Lígia Valéria e Gabriel Pires, pelas valiosas dicas sobre programação em Delphi, programação em Oracle e sites da Internet, respectivamente.

A Adriano Farias e Rodrigo Assad, por sua contribuição na implantação e testes do protótipo na FUNAPE.

A todos os colegas de trabalho que, com a minha ausência, tiveram as suas cargas de trabalho aumentadas.

Ao meu orientador, professor *Francisco Brasileiro*, do Departamento de Sistemas e Computação, Campus II, UFPB, pelo incentivo e pela dedicação do seu tempo.

À minha esposa *Gilka* e aos meus filhos *Giulia* e *Renan* pela paciência e estímulo.

Aos meus pais, que me fizeram ver, desde cedo, o valor da educação e do trabalho.

Sumário

Índice de Figuras

1. Introdução

Nas décadas de 60 e 70, os usuários e administradores de *mainframe* dispunham de técnicas de segurança computacional com boa eficiência para as demandas da época. As instalações baseadas em *mainframe* eram estruturadas em modelos clássicos, com respostas adequadas para as necessidades funcionais e de segurança dos seus sistemas, programadores e usuários. Entretanto, dois fatores transformaram em modelos desatualizados os modelos de segurança adotados naquela época [PFL97]:

- **Uso Pessoal do Computador**: seja para negócios ou lazer, a explosão da computação pessoal atraiu milhões de novos usuários em todo o mundo. Atualmente, aplicações amigáveis (*user friendly*) permitem o uso do computador por pessoas com pouco ou nenhum conhecimento sobre *hardware* ou programação;

- **Sistemas de Rede de Acesso Remoto**: Computadores estão conectados em larga escala, quer seja em número, quer seja em abrangência. Hoje, um usuário tem acesso a informações e serviços que estão, de forma quase que natural e imediata, disponíveis também para inúmeras outras pessoas em todo o mundo através de uma miríade de sistemas computacionais distintos. Um usuário tradicional de mainframe daquela época não podia nem sequer imaginar tal funcionalidade.

As novas frentes de atuação que surgiram associadas com estas características promoveram uma verdadeira revolução na tecnologia da informação e no escopo em que a computação passou a atuar. De forma associada com as novas funcionalidades que foram introduzidas na vida das pessoas, também surgiram novas ameaças e riscos no uso dos computadores. Normalmente, os "usuários finais" não estão conscientes das ameaças envolvidas no uso do computador e, mesmo aqueles com a percepção do problema, normalmente não sabem como agir para reduzir os riscos. Neste contexto de conectividade plena, os requisitos de segurança dos sistemas cresceram bastante, principalmente com o advento das aplicações distribuídas [BIR96].

Alavancadas pela Internet, as aplicações distribuídas estão emergindo de nichos especializados para se transformarem em aplicações mais populares, atingindo os mercados de massa, onde estão os produtos do dia-a-dia das pessoas. De uma forma implícita, a experiência com a Web tem feito a sociedade moderna notar o potencial da computação distribuída. Um dos aspectos que mais contribui para isto é a potencialidade de fazer negócios pela rede, o chamado comércio eletrônico (*e-commerce*). Entretanto, realizar

transações comerciais através da Internet requer um alto nível de segurança das aplicações distribuídas envolvidas.

Aplicações distribuídas estão sempre expostas a ataques e/ou falhas, estejam ou não disponibilizadas através da Internet [BIR96]. Fraudes e erros humanos são comuns de ocorrer e extremamente difíceis de detectar. Os sistemas, frequentemente, ficam expostos a sua ocorrência, pois normalmente eventos assim não são considerados quando os controles de segurança são implementados nas empresas. Além disso, mesmo a segurança tradicionalmente praticada possui falhas e brechas fáceis de serem vencidas e publicamente conhecidas [HUR97], que para serem exploradas necessitam apenas de alguém com um bom motivo e um pouco de determinação.

Com o rápido crescimento da *World Wide Web* e, proporcionalmente, da demanda por comércio eletrônico, a questão de segurança em sistemas distribuídos vem sendo cada vez mais discutida [GS97]. Em muitos aspectos, a confiabilidade de sistemas distribuídos está amarrada com o futuro da *Web* e da tecnologia que está sendo usada para desenvolvê-la.

Cercadas de desafios técnicos a serem vencidos, as técnicas que buscam transações eletrônicas seguras em aplicações distribuídas como um todo, não apenas as de comércio eletrônico, têm as suas fundações baseadas em complexas teorias matemáticas [WAY97]. Em praticamente todos os casos, são usadas variações de criptografia e assinaturas digitais, onde existem senhas e outras informações consideradas privadas, que devem ficar fora do alcance de estranhos. Tais senhas e outras informações que devem ser mantidas secretas, representam a base da maioria dos controles de segurança disponíveis, principalmente para a autenticação das partes envolvidas em uma transação[1]. Normalmente, a segurança do sistema é diretamente proporcional à segurança de tais senhas.

Por tal importância, as senhas representam um dos principais pontos de ataques dos sistemas computacionais e uma das premissas básicas de um sistema de segurança é manter as senhas e outras informações sigilosas longe do alcance de estranhos. Quando falamos de senhas com poucos caracteres, até que é possível guardá-las na memória do usuário, o que, pelo nível de tecnologia atual, talvez seja o local mais inacessível e seguro para isto. Entretanto, com o advento de sistemas de chaves públicas isto ficou impraticável, já que as chaves privadas tratam-se normalmente de longas cadeias de caracteres, quase

[1] Neste documento, consideramos que uma transação é a realização de uma determinada ação por uma aplicação através da troca de mensagens entre componentes obedecendo a um protocolo.

que aleatórias, sendo impossível para um humano memorizá-las. Outro agravante é que as senhas usadas para garantir a autenticação de um usuário, mesmo as não triviais, normalmente são estáticas. Se o fraudador descobre a chave adequada, pode se passar integralmente pelo dono original até ser detectado, o que pode significar muito tempo ou grandes prejuízos.

Existem duas regras básicas de segurança [WHI97]: a primeira, considera que não há segurança perfeita, o máximo que se pode tentar é equilibrar o custo e a complexidade de mecanismos de segurança com os riscos desses mecanismos serem quebrados - o uso da tecnologia adequada pode diminuir os riscos mas sempre com um determinado custo; a outra regra básica considera que o comportamento das pessoas influencia no nível de riscos do sistema ter sua segurança quebrada, isto é, não há tecnologia que seja plenamente eficaz se não houver um comportamento seguro das pessoas. Dessa forma, uma abordagem de segurança bem sucedida deve incentivar o comportamento seguro e, principalmente, deve poder detectar o comportamento não seguro.

O objetivo deste trabalho é propor um mecanismo, chamado de encadeamento de transações, que atua na detecção de quando senhas e outras informações sigilosas vazaram e estão sendo usadas de forma fraudulenta. Atuando na camada de aplicação, o encadeamento de transações permite que seja criado um controle de segurança fim-a-fim entre os componentes de aplicações distribuídas, independentemente do uso ou não de outras técnicas de controle. Ele possui também características ligadas à prevenção de fraudes, pois permite uma autenticação mais eficiente das partes envolvidas em uma transação do que aquela baseada apenas em senhas estáticas. Isto se dá porque o encadeamento de transações propicia uma associação da senha com alguns elementos dinâmicos e de validade temporária. Além disso, o encadeamento de transações também auxilia no controle de erros humanos e falhas de *software* ou *hardware* por possibilitar uma detecção posterior de quando a camada de dados persistente sofreu alterações indesejáveis de origem interna ou externa, com intuitos criminosos ou não.

O restante desse documento está organizado como segue. Inicialmente, no Capítulo 2, faremos uma rápida discussão das classes de problemas relacionados com a segurança computacional que podem permitir vazamento de informação e geração de fraudes. Neste contexto de risco é necessária a utilização de mecanismos de defesa, que também serão abordados no Capítulo 2. No Capítulo 3, vamos enfocar a questão da segurança voltada para aplicações distribuídas, os requisitos para a prática de comércio eletrônico com segurança e o problema do gerenciamento e guarda de senhas e outras informações de caráter privado. No Capítulo 4, discutiremos a necessidade de se manter um controle fim-a-

fim voltado para a segurança, já que o uso de técnicas tradicionais de segurança nem sempre é suficiente para propiciar a segurança necessária. No Capítulo 6, apresentaremos a técnica do encadeamento de transações como uma alternativa de controle adicional e complementar de segurança de fácil implementação e baseado no modelo fim-a-fim e descreveremos a construção de um exemplo completo, baseado na técnica do encadeamento de transações aqui proposta, objetivando melhorar a segurança no relacionamento entre os provedores de acesso à Internet e seus usuários, com vantagens para ambas as partes. Finalmente, no Capítulo 7, analisaremos os resultados obtidos e as vantagens da aplicação da técnica do encadeamento de transações, tanto de forma isolada quanto associada à outras técnicas, para melhorar os níveis de segurança das aplicações. Há ainda o Anexo A que apresenta os fundamentos de sistemas criptográficos.

2. Segurança dos Sistemas Computacionais e suas Ameaças

Quando se fala em roubar um banco, os objetivos mais óbvios talvez sejam dinheiro e jóias, entretanto, a lista de todos os clientes do banco, com os seus respectivos endereços, também pode ser muito útil para um banco concorrente. Esta lista pode tanto estar na forma de um relatório impresso, como representada através de registros magnéticos em um disco ou uma fita ou, ainda, estar contida em uma mensagem sendo transmitida através de uma rede. Enquanto dinheiro e jóias são únicos e podem ser fisicamente isolados, a variedade de formas que pode assumir um alvo digital tornam a segurança computacional extremamente difícil, pois o objetivo de um crime envolvendo computadores pode ser qualquer elemento de um sistema computacional[2].

Alguns aspectos fazem a segurança computacional apresentar algumas diferenças com relação à segurança tradicional, usada, por exemplo, para proteger valores monetários. O "criminoso digital" enfrenta menor dificuldade com o tamanho e a portabilidade dos seus alvos criminosos enquanto que manusear um grande quantidade de cédulas, jóias ou barras de ouro exige uma logística mais elaborada. Cabendo com facilidade em uma pasta, o transporte de alguns *gigabytes* de informações valiosas, por exemplo, é uma tarefa muito mais simples. Outra diferença é que o "criminoso digital" possui a facilidade e a habilidade de evitar o contato físico. Os mesmos recursos de conectividade disponibilizados pela tecnologia e usados para a oferta de serviços legítimos também servem de canal para a prática de atos escusos. O intruso eletrônico pode ser furtivo ao ponto de passar despercebido, enquanto que um ladrão convencional precisa empregar um pouco mais de ações físicas e, mesmo que não seja detectado no momento do crime, o ladrão comum geralmente deixará marcas - no mínimo, a falta do item roubado. "Roubo digital", por sua vez, pode ser meramente uma cópia das informações desejadas. Além destas facilidades para o "crime digital", o roubo tradicional neste contexto também é extremamente vantajoso, pois os bens digitais físicos, como o *hardware*, possuem valores que são, de forma inversamente proporcional às suas dimensões, bastante elevados. Um servidor de aplicações robusto, que pode ser transportado por dois homens, chega a valer várias dezenas de milhares de dólares.

---

[2] Neste documento, um sistema computacional é caracterizado por uma coleção de *hardware*, *software*, mídias de armazenamento, dados e as pessoas envolvidas em tarefas computacionais distribuídas ou não .

Alguns autores comparam a segurança computacional, nos dias de hoje, com uma situação muito próxima do que ocorria nos tempos do velho oeste americano. Enquanto algumas instalações são extremamente protegidas, reconhecendo o valor e a vulnerabilidade de seus dados, outras estão completamente abertas, com graves deficiências no aspecto de segurança.

Cada vez mais a informação vem sendo considerada como um dos principais ativos das empresas. Seja para proteger seus segredos de negócio, suas estratégias comerciais ou na proteção do capital intelectual, a segurança das informações é hoje fator de sobrevivência e competitividade para as corporações modernas, sendo considerada estratégica por 87% dos executivos. Apesar deste alto índice de sensibilização, apurado pela **4ª Pesquisa Nacional sobre Segurança da Informação** [PSI98], realizada em 1998, apenas 67% das empresas possuem uma Política de Segurança formalizada e, na maioria dos casos (69%), está desatualizada, não contemplam todos os ambientes ou não é conhecida pelos usuários.

Os resultados da mesma pesquisa apontam que o problema é grave. Das 176 empresas que participaram da pesquisa, 35% reportaram ter sofrido algum tipo de invasão, sendo que 27% das invasões aconteceram nos últimos 12 meses. Este número é crescente e significa um aumento de 125% em relação ao resultado de 1997. O mais agravante é que 45% das empresas não sabem se foram invadidas. Assim, somente 20% das empresas afirmaram com convicção que nunca sofreram algum tipo de invasão. O número mais alarmante obtido na pesquisa é que a grande maioria não possui uma política de utilização/segurança e somente 22% possuem algum plano de ação formalizado em caso de ataques/invasões (Figura 1).

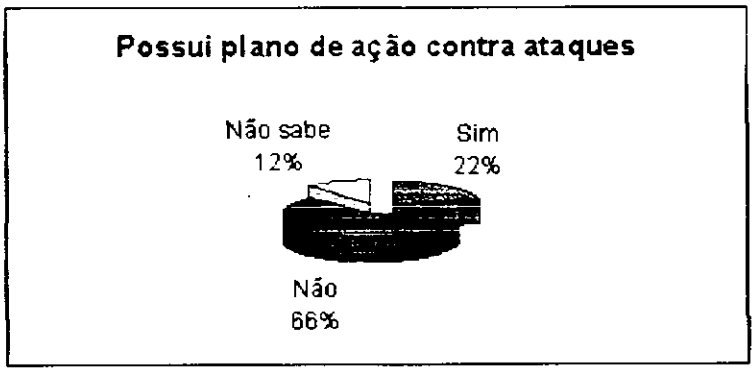

Figura 1: **Empresas que possuem Plano de Ação contra ataques**

Este problema não é específico das empresas brasileiras. Em pesquisa recente realizada pelo *Federal Bureau of Investigation/Computer Security Institute* [TLA98], 74% das empresas americanas tiveram problemas de acesso indevido com seu pessoal interno ou sofreram algum tipo de invasão externa. No relatório de outra pesquisa realizada pela

Information Week [IW97], 73% das empresas americanas reportaram problemas de segurança nos últimos 12 meses.

Outro aspecto que contribui para a falta de informação sobre segurança é que as empresas que sofrem algum tipo de ataque raramente divulgam o fato, quase sempre não instauram processos e, muito menos, levam adiante qualquer investigação policial, com receio que isto possa afetar a sua imagem pública e/ou credibilidade. Outro resultado da 4ª Pesquisa Nacional sobre Segurança da Informação indica que somente em 13% dos casos as empresas recorrem a providências legais. Na maioria das vezes, a atitude tomada em relação aos problemas se limita a providências internas (45%) ou apenas à correção do problema (30%), sendo que em 12% nenhuma providência é realizada (Figura 2).

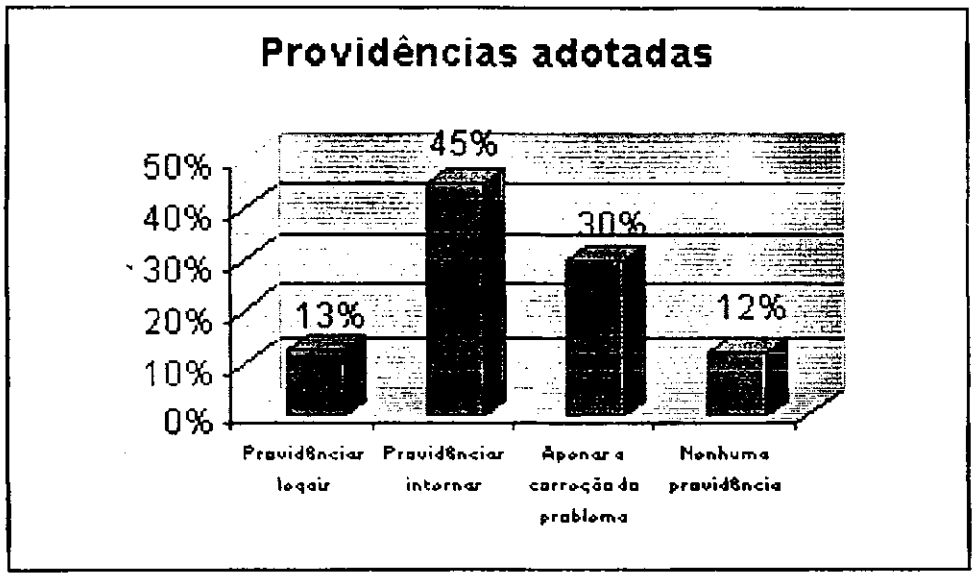

Figura 2: **Providências Adotadas contra Invasões**

Quem se sentiria seguro em confiar as suas economias a um banco, logo após ele ter admitido publicamente a perda de milhões de dólares através de um desfalque por computador? Embora, com certeza, após um fato assim, o banco revisaria e melhoraria substancialmente os seus procedimentos de segurança. Essa é uma forma dolorosa de tomar consciência da necessidade de segurança.

Some-se a isso, o fato de que investigações criminais são baseadas em estatutos que não reconhecem sinais eletromagnéticos como propriedade e os jornais normalmente divulgam invasão de computadores como sendo coisas de adolescentes, um trote, equivalente a pichação de um muro.

Nas seções a seguir, serão apresentados os principais tipos de riscos que estão envolvidos nos sistemas computacionais.

2.1. Conceitos Elementares dos Riscos Computacionais

Em um contexto de riscos, uma **exposição** (*exposure*) é uma forma possível de perda ou dano em um componente de um sistema computacional. Exemplos de exposição são o acesso e/ou divulgação não autorizada de dados, modificação de dados ou impedimento do legítimo acesso aos dados. Uma **vulnerabilidade** (*vulnerability*), por sua vez, é uma fraqueza no sistema de segurança que pode ser explorada para causar perda ou dano. Um **ataque** (*attack*) é a exploração de uma vulnerabilidade. **Ameaças** (*threats*) em sistemas computacionais, são circunstâncias que têm a potencialidade de causar perda ou dano, ou seja, possibilitam ataques. Exemplos de ameaças são tentativas de invasão, erros de operação não intencionais, falhas internas de *hardware*, *bugs* de *software*, desastres naturais etc. Em resumo, a exploração de uma vulnerabilidade por um ataque decorrente de uma ameaça causa uma exposição.

Os três principais componentes de um sistema computacional sujeitos a ataques são *hardware*, *software* e dados. Estes três elementos, e as conexões entre eles, constituem a base das vulnerabilidades da segurança computacional.

Existem ainda os **controles** (*controls*), que são medidas de proteção (uma ação, dispositivo, procedimento ou técnica) que reduzem ou eliminam vulnerabilidades [HIG97]. Nós abordaremos os controles mais adiante. Veremos a seguir classes e exemplos de ameaças

2.2. Classes de Ameaças

Existem quatro classes de ameaças (Figura 3) que potencialmente podem permitir ataques que exploram as vulnerabilidades dos componentes básicos dos sistemas computacionais: **interceptação**, **interrupção**, **modificação** e **fabricação** [PFL97].

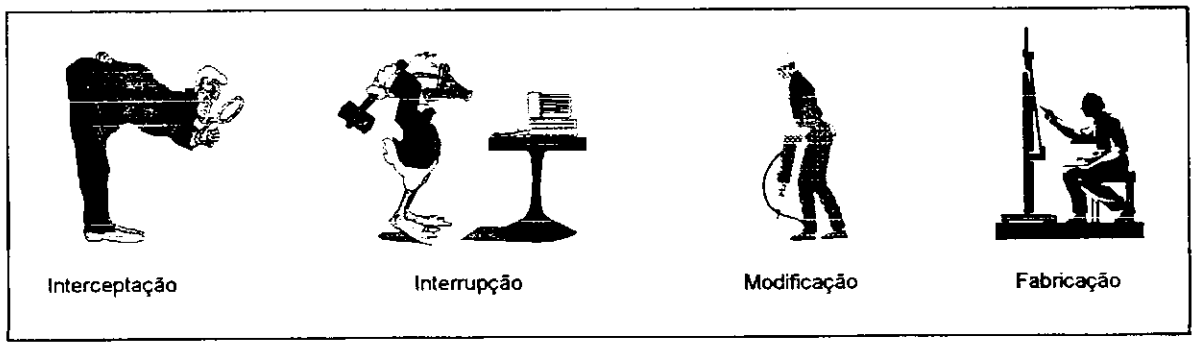

Figura 3: **Classes de Ameaças aos Sistemas Computacional**

Em segurança computacional, quando ocorre uma **interceptação** significa que alguma parte[3] não autorizada obteve acesso a um componente do sistema. Exemplos deste tipo de falha são a cópia ilícita de programas ou arquivos de dados ou uma escuta clandestina para capturar dados em uma rede ou um canal de teleprocessamento. Ao contrário de perda ou modificação, que podem ser descobertas mais rapidamente, uma interceptação normalmente não deixa traços que possam ser facilmente detectados. Nesta classe, podemos agrupar todas as ameaças ilícitas que envolvam o uso ou acesso indevido de componentes.

No mesmo contexto, quando acontece uma **interrupção** é sinal de que um componente do sistema foi perdido ou se tornou indisponível ou não usável (inservível). Exemplos de interrupção são a destruição ou quebra de um dispositivo de *hardware* e a remoção de um arquivo de programa. A classe interrupção agrupa todas as ameaças que culminam com a indisponibilidade permanente de um componente.

No caso de ocorrer uma **modificação**, significa que uma parte não autorizada não somente obteve acesso a um componente mas também o modificou. Exemplos de modificação são a mudança de valores em um banco de dados, alteração de um programa para que ele se comporte de modo diferente da sua especificação ou alteração em dados sendo transmitidos eletronicamente. Embora não seja comum, também é possível a modificação de *hardware*. Alguns casos de modificação são mais facilmente detectados, como a mudança do conteúdo de uma *homepage* institucional. Outras, como a inclusão de computação adicional em um programa, são mais sutis e difíceis de detectar. Nesta classe encontram-se todas as ameaças que produzem a alteração de componentes.

Finalmente, quando uma ocorre uma **fabricação**, é sinal de que alguém ou algo criou uma falsificação de um elemento em um componente de um sistema computacional. O intruso pode adicionar registros em um banco de dados existente ou inserir transações espúrias em uma rede. Algumas destas adições podem ser facilmente identificadas como falsificações, outras se realizadas com habilidade, tornam-se, virtualmente, indistinguíveis das reais. Nesta classe, concentram-se as ameaças que caracterizam-se pela introdução de objetos ilegítimos em um componente.

Estas classes de ameaças descrevem os tipos de exposição possíveis que sofrem os sistemas computacionais, as vulnerabilidades serão discutidas a seguir.

---

[3] Neste documento, usaremos o termo **parte** para descrever algo que pode ser uma pessoa, um programa ou um sistema computacional no contexto da segurança de aplicações.

2.3. Vulnerabilidades

As **vulnerabilidades** são fraquezas no sistema de segurança que podem ser exploradas em ataques, possibilitados pela presença de ameaças, para causar perda ou dano aos sistemas computacionais.

A **Figura 4** ilustra quais as classes de ameaças que se aplicam a cada um dos componentes básicos de um sistema computacional: *hardware*, *software* e dados[4]. Estes três elementos, e as conexões entre eles, são potenciais pontos fracos da segurança. Nas seções seguintes, discutiremos um pouco sobre as vulnerabilidades associadas com cada um deles.

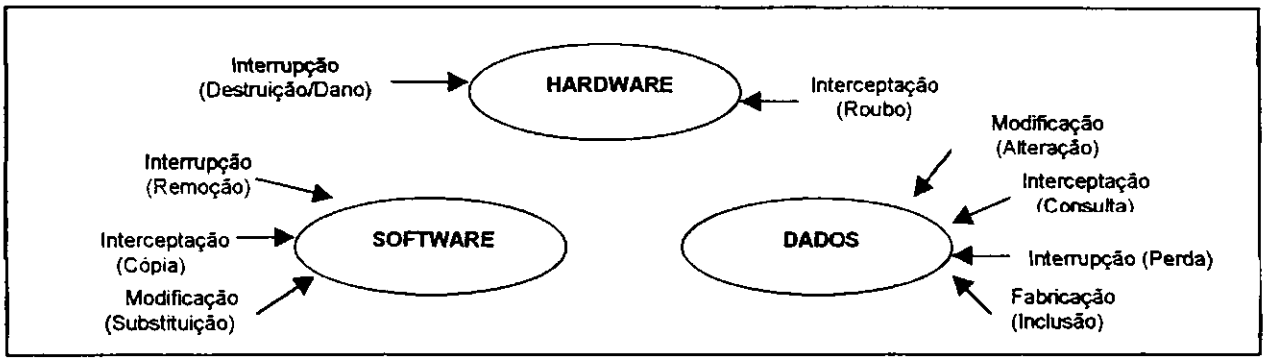

Figura 4: **Vulnerabilidades de Sistemas Computacionais**

2.3.1. Ameaças para o Hardware

Por ser tangível e visível, um dispositivo físico de *hardware* é um ponto de ataque potencial. Computadores podem sofrer com inundações, incêndios e desabamentos. Podem, ainda, ser esbarrados, chutados, derrubados, quebrados etc. É possível que líquidos diversos ou comida sejam derrubados em seu interior. Até poeira e fumaça de cigarros podem danificar componentes de precisão. Todos estes exemplos, capazes de causar danos sérios ao *hardware* envolvido, fazem parte da categoria das ameaças involuntárias, ou seja, atos acidentais não intencionais.

Existem ataques mais graves, agrupados na categoria das ameaças voluntárias, nos quais alguém, deliberadamente, deseja danificar um componente de *hardware*. Intencionalmente,

---

[4] Embora *hardware*, *software* e os dados sejam os principais pontos de ataque a sistemas computacionais, existem outros elementos importantes e também vulneráveis. Itens como as mídias usadas para backup, as redes de comunicação e as pessoas também devem ser considerados como pontos de ataque.

pessoas podem destruir/danificar (**interrupção**) ou roubar (**interceptação**) elementos de *hardware*. Desde bombas a incêndios criminosos, passando por armas e ferramentas, até o uso de lápis, chaves e outros objetos mais comuns, praticamente tudo pode ser usado para violação ou dano. A lista das formas possíveis de ataques humanos intencionais a computadores é, virtualmente, infinita, pois não há muitos requisitos técnicos necessários para se realizar uma agressão deste tipo. De todas as maneiras que as pessoas podem, deliberadamente, atacar equipamentos de *hardware* para limitar a sua disponibilidade, roubo e destruição são as técnicas mais usadas.

A vulnerabilidade do *hardware* já é reconhecida de longa data e dispositivos de proteção sempre foram comuns em CPDs centralizados. Entretanto, com a proliferação dos microcomputadores, este trabalho de defesa ficou mais complexo pois elementos de *hardware* valiosos passaram a ficar desprotegidos em cima de mesas de trabalho. Embora, geralmente, os servidores de arquivos e aplicação ainda fiquem relativamente protegidos em áreas mais isoladas, as estações de trabalho, impressoras e equipamentos de conectividade normalmente estão distribuídas e suscetíveis aos mais diversos atos de vandalismo.

2.3.2. Ameaças para o Software

O *hardware* é inútil para o usuário sem o *software* (sistema operacional, utilitários e aplicações). O *software* pode, intencionalmente ou acidentalmente, ser destruído, modificado ou extraviado. Independente do motivo, o resultado é o mesmo: perda de disponibilidade de serviços. Um ataque ao *software* é detectado, normalmente, quando alguém tenta executá-lo, mas também é possível passar despercebido, pois enquanto um ataque ao *hardware* geralmente deixa algum tipo de vestígio, modificações em algumas linhas de código fonte não deixarão nenhuma marca facilmente visível. Principalmente, se o atacante após a compilação restaurar o código fonte original. Assim, é muito difícil detectar **que** o software foi modificado e, mais ainda, **onde** o *software* foi modificado.

As principais ameaças para o *software* são:

- **Interrupção (remoção)**: é relativamente fácil remover *software*. Intencionalmente, basta possuir os direitos de acesso adequados, pois em praticamente todos os sistemas operacionais os comandos para manusear arquivos de programas e arquivos de dados são os mesmos. Acidentalmente, onde os direitos de acesso são legítimos e necessários para o trabalho normal, os casos de perda de *software* são inúmeros e corriqueiros. Praticamente todos os profissionais envolvidos com computação já se depararam com alguma situação deste tipo ou, no mínimo, estão suscetíveis a sua ocorrência. Desde o administrador que remove acidentalmente um diretório, passando pelo operador que

restaura incorretamente um *backup* até um programador que remove, sem querer, um arquivo de programa ou salva uma versão desatualizada, sobrepondo a versão correta.

- **Modificação (Substituição)**: neste tipo de ataque, um componente de *software* é modificado para falhar durante a execução ou para realizar uma tarefa não especificada ou adicional. Inserir falhas em um *software* é relativamente simples, bastando trocar alguns *bits* do executável. Dependendo do local da mudança, o *software* pode falhar logo no início da execução ou executar corretamente durante algum tempo antes de ocorrer o problema. Mudanças mais elaboradas buscam produzir um *software* que execute bem na maioria das vezes e que só apresente o comportamento anormal em algumas circunstâncias especiais. Neste caso, temos o que é chamado de bomba lógica (*logic bomb*) [BLU94]. Um exemplo de bomba lógica é o seguinte: um programador, descontente com a demissão e durante o período de aviso prévio, pode inserir nos programas da empresa algumas instruções para, alguns meses após a sua saída, falhar durante a execução ou realizar cálculos incorretos. Outro tipo de mudança pode ser mais sutil, não alterando a funcionalidade normal do programa mas, aumentando-a com algum processamento adicional. Por exemplo, um lote de comando (*script)* usado para automatizar a realização de cópias de segurança (*backups*) pode ser alterado para que também modifique as permissões de acesso a arquivos se for executado em determinado dia da semana ou hora do dia.

  Dentre os tipos de modificação de *software* possíveis, podemos destacar as seguintes categorias:

  - **Cavalo de Tróia** (*trojan horse*): é um programa que, publicamente e oficialmente, realiza uma determinada ação útil enquanto que, por trás (*background*), realiza outra ou outras ações danosas ou não desejadas;

  - **Vírus** (*virus*): é um tipo especial de cavalo de tróia, que possui a capacidade de se auto-reproduzir e que pode ser usado para propagar a "infecção" com um programa modificado de um computador para outro. Por agirem de forma indiscriminada e sem controle, os vírus talvez representem a mais conhecida e danosa das manifestações da modificação maliciosa de *software*. Existe toda uma indústria paralela[5] no

[5] Existem inúmeras versões sobre a origem e motivação para o desenvolvimento de vírus. Algumas atribuem a introdução dos primeiros vírus às empresas produtoras de *software*, numa forma de diminuir a pirataria. Há quem acredite também que, atualmente, os vírus são liberados pelas próprias empresas que desenvolvem e comercializam mecanismos antivírus.

mercado de computação voltada para o controle dos vírus com uma dinâmica constante: diariamente surgem novos tipos de vírus que demandam o desenvolvimento de novas "vacinas";

- **Alçapão** (*trapdoor*): é um programa que possui uma porta de entrada secreta. Por várias razões, muitos alçapões são construídos durante o período de desenvolvimento de um software para auxiliar ou agilizar alguns processos. A não retirada deles da versão final, produz falhas graves na segurança quando, por algum motivo, ficam públicos[6], pegando de surpresa toda a base instalada;
- **Vazamento de Dados** (*information leaks*): são tipos especiais de alçapões voltados exclusivamente para tornar informação acessível a pessoas e/ou programas não autorizados.

Também é plenamente possível desenvolver um novo programa e instalá-lo em um ambiente computacional[7]. Controles inadequados sobre que programas estão instalados e/ou executam em uma determinada máquina permitem este tipo de ataque.

- **Interceptação (Cópia):** Este tipo de ataque inclui a cópia ilegal de *software* ("pirataria"). Poucos mecanismos são plenamente eficazes para impedir, de forma efetiva, a pirataria de *software*. O uso de chaves de ativação é burlado facilmente através da publicação de *sites* na Internet (Ex.: http://www.hackerborg.dk/serial.htm) com relações extensas de números e senhas para os mais diversos programas comercias e, aqueles produtos que utilizam controles mais robustos, com o uso de *hardware* para proteção, são criticados pelos transtornos que causam aos usuários legítimos.

2.3.3. Ameaças para os Dados

Devido a sua visibilidade natural, os ataque aos dados são mais difundidos e graves do que ataques a *hardware* e *software*, porque um número maior de pessoas sabe como usar ou interpretar os dados. De todos os componentes de um sistema computacional, os dados são também os que mais sofrem ataques intencionais [PLF97].

---

[6] Os *softwares* de telefones celulares são célebres pela existência de alçapões, que permitem desde a realização de ligações sem tarifação até fazer escuta clandestina.

[7] Alguns sistemas operacionais, como o UNIX, já trazem nativamente ferramentas de desenvolvimento, disponibilizadas para uso público desde a instalação, o que facilita a ação de intrusos, que podem desenvolver com rapidez programas nocivos.

A **interrupção** ou perda de dados talvez seja o mais comum dos ataques sofridos pelos dados, talvez porque possam ser causados tanto de forma intencional quanto por causas naturais. Dados podem ser perdidos tanto por falha de *hardware*, por erros humanos e também por ações criminosas. Por não terem um valor intrínseco, é difícil mensurar quanto valem dados perdidos. Entretanto, como os dados têm um custo, o custo da perda é mensurável através dos custos envolvidos na sua reconstrução ou para desenvolvê-los novamente, em casos extremos.

A **interceptação** de dados também é uma das ameaças aos dados mais exploradas e pode se manifestar de várias formas, todas com efeitos igualmente danosos. Um exemplo de manifestação é o vazamento de dados confidenciais de uma empresa, o que pode limitar a sua competitividade se estes forem repassados para um concorrente. Outro exemplo é a quebra do sigilo bancário de um cliente, fato que pode causar danos irreparáveis à imagem de uma instituição financeira. Por serem a forma mais usada de exibição de resultados computacionais, os dados contidos em relatórios impressos podem ser facilmente interpretados e entendidos pelo público em geral, sendo a sua captura a maneira mais usual de roubo de informações. A cópia de arquivos e a captura de mensagens em uma rede também são formas usadas para a interceptação de dados.

Os dados também são especialmente vulneráveis à **modificação**. Mudanças sutis e habilidosas nos dados são muito difíceis de se detectar. Um exemplo clássico é o **ataque salame** (*salami attack*), que consiste em retirar alguns centavos das contas de inúmeros clientes, agrupando-os em uma determinada conta corrente, de forma análoga aos fragmentos de carne que são agrupados para formar um salame. O valor final, dependendo do tempo e da quantidade de contas envolvidas, pode ser extremamente alto. O fraudador que realiza este tipo de ataque baseia-se na certeza de que, dificilmente, um cliente comum percebe o problema, já que trata-se de um valor muito pequeno individualmente. Mesmo que isto ocorra e o cliente avise ao banco, o funcionário com quem ele terá contato provavelmente não perceberá que se trata de uma fraude e tentará minimizar a perda, atribuindo a diferença a algum processo de arredondamento.

Embora exigindo um processo mais elaborado, a **fabricação** de dados também é possível. Uma forma de fabricação muito comum é a geração de transações de comércio eletrônico ilegítimas, a partir da captura de senhas e números de cartões de crédito de outras pessoas. Outra forma de ataque pode ser praticada interceptando-se mensagens de comunicações bancárias. Considerando a captura de uma transferência de valores entre contas, o fabricante pode reenviar a transação (*replay attack*), causando a duplicação da operação ou tentar modificar o seu conteúdo, indicando outra conta destino ou outro valor.

2.4. Criminosos Digitais

Quando os ataques cometidos contra sistemas computacionais são intencionais e praticados por pessoas, temos uma categoria especial de ataques classificada como crimes computacionais[8] ou digitais. O protótipo do mocinho e bandido vastamente difundido no cinema, em que o bandido é mal encarado, sujo e mal vestido enquanto que o xerife possui uma boa figura e é conhecido e respeitado por todos nem sempre é válido aqui. Alguns criminosos computacionais podem até ser tipos sinistros, mas, a maioria, está dentro da normalidade de aparência e comportamento esperado. Vestem-se bem, possuem formação acadêmica, são respeitados em seus grupos. Podem ser adolescentes, estudantes ou executivos de meia-idade. A motivação para cometer tais crimes também é variada, incluindo desde pessoas comuns tentadas por uma chance de obter ganhos pessoais até aquelas motivadas pelo intuito de cometer uma vingança. Há, ainda, as pessoas que estão buscando vencer um desafio ou, simplesmente, são levadas pela curiosidade. No outro extremo, temos pessoas mentalmente doentes, gratuitamente hostis ou radicais de alguma causa, que cometem os ataques computacionais como um símbolo ou protesto. Independentemente da aparência ou motivação, às pessoas que cometem tais crimes podem ser classificadas em três grandes grupos [PFL97]: amadores, *hackers*[9] [LEE96] e criminosos profissionais.

2.4.1. Amadores

A maioria dos crimes computacionais cometidos reportados até hoje foram cometidos por amadores. A maioria dos desfalques são cometidos por pessoas de comportamento absolutamente normal até o momento em que descobrem uma falha na segurança de sistemas que possibilite que elas obtenham dinheiro ou outras vantagens. Na maior parte das vezes, são usuários, analistas, programadores, técnicos ou operadores, fazendo suas tarefas normais, quando são postos frente a frente com uma situação em que podem levar alguma vantagem, quer seja com o furto de algum *hardware*, com a divulgação indevida de alguma informação confidencial ou até pela modificação ou fabricação de informações nos sistemas que operam. Normalmente, a "carreira" do amador começa com coisas ínfimas,

---

[8] Consideramos crime computacional qualquer crime envolvendo ou auxiliado pelo uso de componentes computacionais.

como usar o computador e a impressora da empresa para escrever cartas ou convites pessoais, manter tabelas de campeonatos de algum esporte, produzir cartelas de bingo e assim por diante. Quando não há objeções, a situação pode se expandir ao ponto do amador passar a fazer pequenos "bicos" dentro do seu expediente e usando os recursos computacionais da empresa. Outros ataques mais destrutivos, podem ser voltados contra a própria empresa, motivados por subversão, vingança ou ganância: desde a remoção de arquivos e programas por um funcionário insatisfeito, passando pela alteração de valores usando as próprias aplicações da empresa, chegando até à divulgação de informações confidenciais para concorrentes. Os ataques cometidos por pessoas internas representam a maioria das fraudes e roubos reportados e representam um problema grave e real da segurança computacional. Segundo a 4ª Pesquisa Nacional sobre Segurança da Informação, no aspecto de identificação dos responsáveis por quebra de segurança, 34% dos problemas ocorrem com empregados, 7% com clientes, 14% com outros como fornecedores, prestadores de serviço, concorrentes, estagiários e governos estrangeiros, 24% com *hackers* e invasores e 21% tem causa desconhecida (Figura 5). Na mesma pesquisa e ainda especificamente em relação a invasões e acessos indevidos, até o ano passado as ameaças internas representavam cerca de 67% dos problemas, passando este ano para 46%. Mas, longe de significar que os problemas internos diminuíram, o que aconteceu é que os problemas com ataques e invasões externas dispararam, sendo a Internet hoje apontada por 42% das empresas como o principal ponto de risco externo. O risco também é crítico no ambiente de Intranet. O modelo atual para segurança das redes tem assumido que o "inimigo" está do lado de fora da empresa enquanto que dentro todos são confiáveis. Esta idéia tem feito que os administradores de rede utilizem uma estratégia de segurança que restringe o acesso para qualquer usuário externo e, por outro lado, libera de forma irrestrita o acesso aos servidores para a totalidade dos usuários internos, numa estratégia simples mas inadequada.

---

[9] Termo usado normalmente para se referir a pessoas que invadem computadores violando a segurança, chamados também por: *crackers*, "*bad guys*", intrusos, vândalos ou, simplesmente, criminosos.

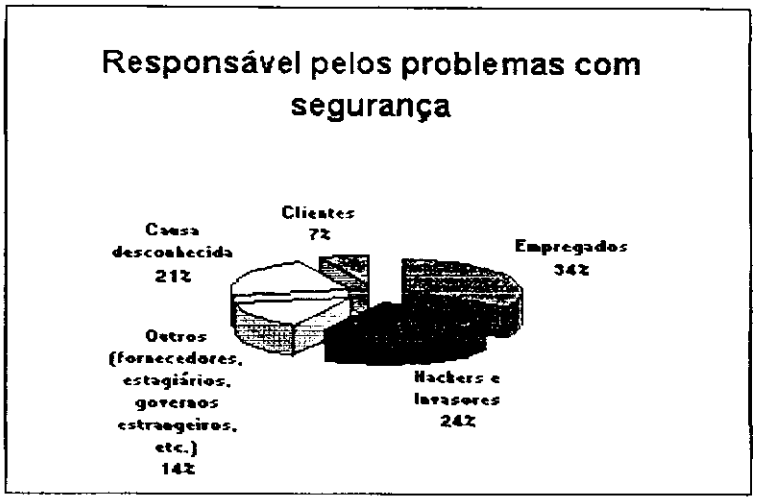

Figura 5: **Estatística sobre Responsáveis por Problemas de Segurança**

2.4.2. Hackers/Crackers

Os *crackers* ou *hackers* de sistemas geralmente são estudantes que, de forma geral, apenas tentam acessar componentes computacionais para os quais não têm a devida autorização [FAB97]. A "quebra" da segurança computacional por *hackers* é considerada, por eles, como um crime sem vítima. Eles acreditam que ninguém é machucado ou posto em perigo pelo roubo de um pouco de tempo de um computador, ou pela tentativa de se conectar em uma máquina, apenas para verificar se é possível fazê-lo. Na ausência de advertências explícitas sobre a proibição de ultrapassar um determinado ponto de um sistema, os *hackers* supõem que o acesso é permitido. Existe uma rede clandestina usada para a troca de informações, senhas, dicas e técnicas de invasão entre *hackers* [GUR97]. Motivados pela vaidade, os *hackers* são agilíssimos na divulgação de uma invasão para garantirem a primazia do feito. Desta forma, antes que o administrador do sistema se dê conta do ocorrido, milhares de outros *hackers* já estão repetindo a invasão, transformando um fato isolado em um incidente de grandes proporções.

Não há uma motivação única ou comportamento padrão para os ataques. Existem *hackers* que buscam auto-satisfação ou notoriedade, outros estão atrás de ganhos e vantagens pessoais enquanto outros preferem simplesmente causar dano, perda ou produzir confusão. Entretanto, quanto mais segura uma instituição diz ser, mais atraente ela se torna para os *hackers* e mais comemorada é uma invasão. Desde orgãos governamentais, passando por instituições militares ou de investigação até universidades, ninguém está a salvo.

A ação dos *hackers* é muito grave e tem causado grandes prejuízos por todo o mundo, principalmente pelos transtornos e pela exposição a que submetem as empresas. Quando apanhados, os *hackers* são indiciados seriamente e alguns já sofreram punições muito

severas. Apesar disso, invadir computadores continua sendo um crime atraente, principalmente entre os jovens.

2.4.3. Profissionais

Os criminosos computacionais profissionais, por sua vez, conhecem muito bem os objetivos do crime computacional, buscando sempre alguma espécie de lucro pessoal. De forma diferente dos *hackers* e amadores, com motivações variadas, os criminosos profissionais concentram a sua atuação onde o ataque produza, de uma forma ou de outra, alguma vantagem e, se existirem brechas na segurança e menos desafio, melhor. Normalmente, eles começam como profissionais de computação normais e se engajam no crime motivados pelos ganhos e pelas perspectivas de impunidade. Embora existam indícios de que o crime organizado e grupos internacionais estão se engajando no crime digital, é improvável que um ladrão de carros, um assaltante de bancos ou um assassino mude a sua área de atuação e passe a atuar na área computacional, que exige uma qualificação específica.

Como citado anteriormente, algumas empresas continuam sendo reticentes em abrir inquéritos contra criminosos digitais. Na verdade, após descobrirem que foram atacadas ou fraudadas, as empresas ficam, de certa forma, aliviadas se o criminoso, ao contrário dos *hackers*, for discreto. Além disso, na maioria dos casos a empresa passa a concentrar os seus esforços em aumentar o nível de proteção dos seus recursos, para evitar uma nova ocorrência do incidente, ao invés de reunir evidências que possam levar à identificação ou captura do fraudador, o que é compreensível. Desta forma, o criminoso digital continua livre para atacar outra empresa.

2.5. Exemplos de Ameaças

Descrever com exatidão o patrimônio computacional de uma empresa não é uma tarefa trivial. Englobando bens tangíveis e bens intangíveis, é difícil determinar qual o prejuízo gerado pela indisponibilidade de uma informação ou de um serviço, ou quanto custa reconstruir um cadastro de clientes ou um banco de dados de contas a receber. Como exemplos de bens tangíveis, podemos citar computadores, dados proprietários, *backups* e arquivos, manuais e livros, relatórios, *software* comercial, equipamentos de comunicação e cabeamento, registros pessoais, registros de auditoria etc [OLI94] [MER98]. Na categoria dos bens intangíveis, podemos incluir a segurança e saúde dos funcionários, a privacidade dos usuários e clientes, senhas pessoais, a reputação e imagem pública da empresa, a satisfação dos clientes, a continuidade ou disponibilidade do processamento etc. Cotidianamente, tais bens estão vulneráveis a ataques possibilitados por ameaças, que nem

sempre são, explicitamente, ligados à informática em si. Como exemplos de ameaças comuns estão: doenças de pessoas-chave[10], doença simultânea de muitas pessoas (epidemia de gripe), perda (demissão, morte) de pessoas-chave, perda de serviços telefônicos ou de rede, perda de serviços básicos (água, luz, telefone) por um curto período, perda de serviços básicos (água, luz, telefone) por um longo período, queda de raios, inundação, roubo ou perda de fitas e discos magnéticos, roubo ou perda do *laptop* de pessoas-chave, roubo do computador pessoal de pessoas-chave, introdução de vírus, falência de fornecedores de *hardware* ou *software*, falhas (*bugs*) no *software*, empregados subvertidos, funcionários de terceiros subvertidos (por exemplo, manutenção de *hardware*), terrorismo político, greves, invasão por *hackers*, vazamento de informação confidencial ou proprietária etc. Também segundo a 4ª Pesquisa Nacional sobre Segurança da Informação, as preocupações com *hackers* (36%), a espionagem industrial (14%), roubo e furto (28%), acesso indevido (41%), funcionário insatisfeito (61%) e roubo de senhas (30%) foram as ameaças que mais cresceram em relação à pesquisa anterior de 1997 (Figura 6).

Nos EUA, a *American Society for Industrial Security* estima que as companhias americanas perdem anualmente mais de US$ 300 bilhões de dólares com crimes por computador. As empresas de tecnologia e indústrias são as mais atingidas. Segundo relatório do FBI/CSI 1998 [CSI98] [TLA98], o valor médio das perdas anuais é de US$ 568 mil por empresa. Os principais fatores considerados para este cálculo são decorrentes de prejuízos diretos como perda de contratos, roubo de segredos industriais, fraudes financeiras, danos à imagem e custos com investigações, parada de serviços e reposição.

É extremamente difícil mensurar em valores monetários, para uma apólice de seguros por exemplo, todo o conjunto de bens tangíveis e intangíveis. Mesmo que chegássemos a uma determinada quantia de dinheiro que representasse todo o histórico computacional de uma empresa, reconstruí-lo fielmente a partir do zero seria praticamente impossível. Face a este risco, a adoção de mecanismos que permitam reduzir as vulnerabilidades e minimizar o efeito de ataques é fundamental.

[10] Pessoas-chave são aqueles elementos de importância fundamental dentro das empresas, principalmente no aspecto técnico. Exemplos de pessoas-chave são gerentes de CPD's, analistas de suporte, administradores de redes, analistas ou programadores de sistemas de missão crítica etc.

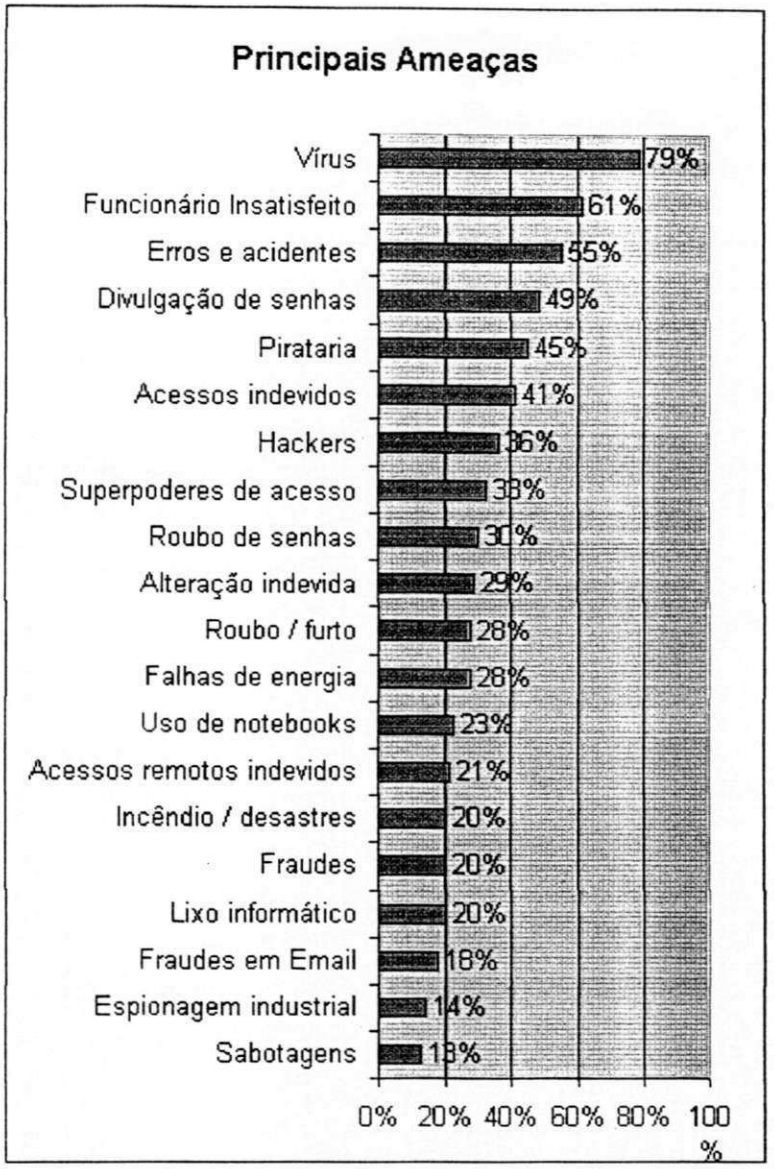

Figura 6: **Estatística sobre Ameaças Computacionais**[11]

2.6. Segurança Computacional

O conhecimento das ameaças potenciais, das vulnerabilidades e dos tipos de ataque a que os sistemas computacionais estão expostos não é suficiente para eliminar o risco de uma perda ou dano. O risco não pode ser eliminado totalmente. No máximo, pode ser identificado

[11] As empresas consultadas escolheram em uma lista de múltiplas respostas as ameaças mais graves. Na consolidação, **vírus** estava presente em 79% das respostas, **funcionários insatisfeitos** em 61% e assim por diante.

e controlado. Os riscos associados com sistemas computacionais certamente vão continuar existindo, como também criminosos digitais motivados.

Neste contexto, o objetivo da segurança computacional é implementar controles que preservem determinadas características dos componentes de um sistema. Tais controles, em alguns casos, são eficazes para evitar ataques, em outros, são capazes apenas de identificar uma determinada vulnerabilidade ou detectar se uma fraude foi praticada. Existe, na literatura sobre o assunto, um certo consenso de que segurança absoluta é impossível e que o conceito de **confiabilidade** (*trust*) é o mais próximo disso que pode ser atingido. Desta forma, podemos dizer que um computador é "seguro" se você pode confiar nele (*hardware* e *software*) para se comportar como você espera ou deseja [GS96], ou pelo menos, se a probabilidade dele se comportar como você espera ou deseja é suficientemente alta.

2.7. Objetivos da Segurança Computacional

A segurança computacional consiste em garantir três características para os componentes básicos de um sistema: **confidencialidade**, **integridade** e **disponibilidade.**

Garantir a **confidencialidade** significa que os recursos são acessíveis somente por partes autorizadas. O acesso aqui discutido é do tipo para leitura (*read-type*): ler, visualizar, imprimir ou, simplesmente, tomar conhecimento da existência do objeto. É a propriedade da segurança melhor entendida porque o seu significado é mais específico e porque existem bons exemplos de preservação da confidencialidade no mundo real.

Pela propriedade da **integridade**, os recursos podem ser modificados apenas por partes autorizadas e/ou somente através de formas autorizadas. Aqui, modificação pode ser: escrever, trocar, mudar o estado, apagar e criar. A **integridade**, ao contrário da **confidencialidade**, não possui um significado único, podendo significar diferentes coisas em diferentes contextos: precisão, acurácia, não modificação, modificação somente através de formas aceitáveis, modificação somente por pessoas autorizadas, modificação somente por processos autorizados, consistência ou produção de resultados corretos e significativos.

Pela característica da **disponibilidade**, os recursos estão (devem estar) acessíveis para quem de direito, ou seja, alguém devidamente autorizado não deve ser impedido de acessar objetos para os quais dispõe de acesso legítimo. Tão complexa de definir quanto a **integridade**, a **disponibilidade** aplica-se tanto a dados quanto a serviços computacionais[12]. Algumas expectativas relacionadas com a disponibilidade são: a presença do objeto ou serviço em uma forma usável, a capacidade de satisfazer as necessidades do serviço, o progresso garantido (tempo finito de espera) e adequado tempo de resposta ou pontualidade (*timeliness)* do serviço. Outra forma de analisar o que é disponibilidade é a partir de alguns dos seus objetivos: resposta oportuna, alocação regular, tolerância a falhas, usabilidade (pode ser usado quando desejado) e concorrência controlada (suporte para acesso simultâneo, gerenciamento de impasses (*deadlocks)* e acesso exclusivo, se desejado).

Embora bastante independentes, estas três qualidades também podem se sobrepor ou serem mutuamente exclusivas. Por exemplo, uma proteção muito forte objetivando a confidencialidade pode restringir severamente a disponibilidade. Por outro lado, um controle de acesso pequeno e centralizado, embora fundamental para preservação da confidencialidade e integridade, não significa, a princípio, um incremento na disponibilidade. Para ilustrar, vejamos como as três propriedades básicas da segurança computacional podem ser aplicadas aos dados (Figura 7).

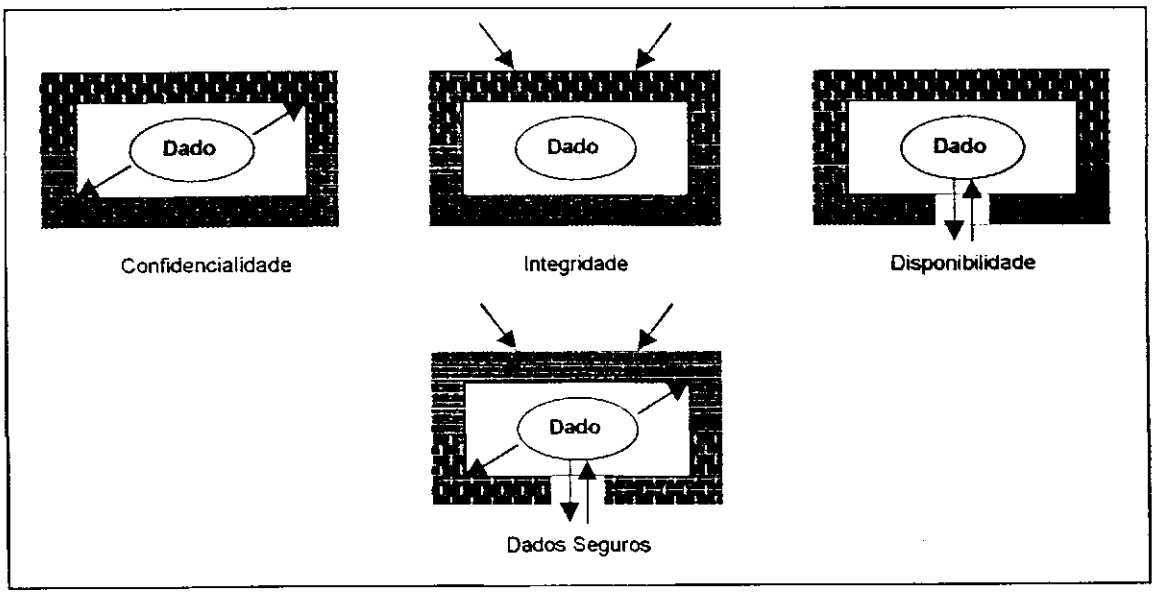

Figura 7: **Segurança dos Dados**

[12] Neste documento, consideramos serviço computacional como o acesso a recursos computacionais.

Na âmbito da pesquisa sobre segurança, as propostas mais bem sucedidas estão concentradas na área de confidencialidade e integridade. Com relação a disponibilidade, ainda estamos no início do entendimento das suas implicações e de como realmente garanti-la.

Além dos aspectos da **confidencialidade**, **integridade** e **disponibilidade**, discutidos anteriormente, outros aspectos também são importantes na composição da segurança computacional:

- **consistência**: garantir que o sistema se comportará como esperado pelos usuários autorizados. Os efeitos são extremamente graves se o *hardware* ou *software* passam a se comportar, abruptamente, de uma maneira diferente da usual, que podem ocorrer após uma atualização de versão (*upgrade*), após a correção de um problema (*bug fix*) ou, mesmo, após a ação de um intruso. Imagine o que aconteceria se o comando do unix *ls* fosse maliciosamente alterado para remover arquivos ao invés de listá-los;

- **controle**: regulamentação do acesso para evitar que pessoas (ou *software*) desconhecidas ou não autorizadas sejam encontradas no seu sistema. Nestas ocorrências, a preocupação passa a ser **quem** fez, **o que** fez e **como** foi feito o ataque. A recuperação nestes casos pode exigir um tempo considerável não só para reconstruir ou reinstalar o seu sistema, em casos extremos, mas também para verificar se alguma informação relevante foi alterada ou divulgada, ou mesmo, para se chegar a conclusão de que nada aconteceu;

- **auditoria**: não apenas intrusos podem causar danos, usuários autorizados também podem cometer erros ou danos intencionais. Em ambos os casos é necessário dispor de uma forma de detectar ou corrigir o que foi feito. Uma maneira de se obter tais resultados é através da manutenção de um registro incorruptível da atividade do sistema, chamado de trilha de auditoria (*audit trail*), que permita a identificação precisa sobre cada ação realizada com o seu respectivo autor. Em algumas aplicações críticas, a trilha de auditoria pode ser poderosa a ponto de permitir uma forma de desfazer (*undo*) transações para restaurar o sistema até um estado correto.

Embora todos estes aspectos acima sejam importantes *per si*, a relevância de cada aspecto com relação a outro é vista de forma diferente, de acordo com o contexto da aplicação. Por exemplo:

- **Aplicações na área financeira:** os aspectos de integridade e auditoria são prioritários, em seguida, estão os aspectos da confidencialidade e disponibilidade;

- **Aplicações na área militar:** o aspecto da confidencialidade é extremamente crítico, depois aparece o aspecto da disponibilidade;
- **Aplicações na área acadêmica:** a integridade é o aspecto fundamental, seguido pelo aspecto da disponibilidade.

2.8. Princípios da Segurança Computacional

Três princípios afetam a direção dos trabalhos em segurança computacional [PFL97]:

> **Princípio da Penetração mais Fácil** (*Principle of Easiest Penetration*)
>
> Pode-se esperar que um intruso vai usar qualquer meio disponível de penetração. Não quer dizer, necessariamente, que ele vai usar a forma mais óbvia e muito menos aquela na qual as defesas mais sólidas foram instaladas.

Por este princípio, o especialista em segurança deve considerar todos os meios possíveis de penetração, porque o reforço em apenas alguns deles, os mais óbvios, pode tornar outros meios, menos naturais, mais atraentes para intrusos. Por exemplo, uma instalação extremamente bem guardada em uma sala forte com sistemas sofisticados de controle de acesso físico e lógico pode ficar exposta pela existência de uma linha telefônica e um modem, usados para suporte e manutenção remota, por exemplo. Por essa filosofia, o **ponto fraco** é a mais grave vulnerabilidade de qualquer sistema de segurança. Em resumo, um ladrão que deseja roubar algo de sua casa não vai preferir arrombar uma porta de madeira maciça de uma polegada se existe uma janela aberta.

> **Princípio da Proteção Adequada** (*Principle of Adequate Protection*):
>
> Bens computacionais devem ser protegidos somente até eles perderem o seu valor. Eles devem ser protegidos em um grau consistente (proporcional) com o seu valor.

Este princípio prega que coisas que têm uma vida curta podem ser protegidos por medidas de segurança que são eficazes por apenas um curto período. Ele aplica-se principalmente a dados, já que *hardware* e *software* têm uma vida útil relativamente longa. O valor de um dado pode ser extremamente alto, mas alguns tipos de dados são interessantes por somente um curto período de tempo. Como exemplo disso temos as informações fornecidas pelo Governo sobre a economia nacional, como taxas de juros que serão praticadas, desvalorização da moeda etc. O conhecimento prévio de tais dados pode permitir que alguém tome vantagem dos efeitos que tais medidas ou informações terão no mercado

financeiro. Suponhamos então que um determinado analista do Governo preparou tais dados 24 horas antes da divulgação oficial e deseja transmiti-los para outro analista para uma verificação final. Um esquema de proteção que exija do atacante mais de 24 horas para quebrá-lo é adequado neste caso, pois após a divulgação não há mais necessidade de confidencialidade.

> **Princípio da Eficácia** (*Principle of Effectiveness*):
>
> Controles devem ser usados de forma eficaz. Eles devem ser eficientes, fáceis de usar e adequados.

Este princípio implica em que os controles voltados para a segurança computacional devem ser eficientes o bastante, em termos de tempo, quantidade de memória, atividade humana ou outros recursos usados, de tal forma, que o uso do controle não afete significativamente a tarefa ou recurso a ser protegido. Além disso, os controles devem ser seletivos para não impedir acessos legítimos.

2.9. Controles

Veremos a seguir os controles ou contramedidas que buscam evitar a exploração das vulnerabilidades de sistemas computacionais.

2.9.1. Criptografia

O mais poderoso e eficaz mecanismo disponível para segurança computacional é a criptografia ou codificação[13]. Através da transformação de dados da sua forma natural em sequências não inteligíveis para pessoas indesejáveis, os profissionais de segurança podem anular o efeito de uma interceptação e impossibilitar a modificação e fabricação de dados. Além da confidencialidade intrínseca e natural, o uso da criptografia também pode ser usado para garantir integridade, porque dados que não podem ser lidos também não podem ser modificados de forma consistente. A criptografia, pelas suas características, é a base dos métodos e protocolos que visam atingir os objetivos da segurança computacional, notadamente a confidencialidade e a integridade.

Embora seja uma das mais importantes ferramentas disponíveis, a criptografia não resolve, sozinha, todos os problemas da segurança computacional. Além disso, se não aplicada

---

[13] Ao longo deste texto, usaremos codificação, encriptação, cifragem como termos relacionados com criptografia.

corretamente, ela pode não ser eficiente na segurança ou degradar a performance total do sistema. Criptografia inadequada talvez seja pior do que criptografia nenhuma porque pode produzir uma falsa sensação de segurança. A técnica de criptografia é tratada com mais detalhes no Anexo A.

2.9.2. Controles por *Software*

Embora nem sempre vistos assim, os programas são elementos importantes na implementação da segurança computacional e devem ser seguros o suficiente para revidar ataques externos. Os programas precisam ser desenvolvidos e mantidos visando satisfazer as demandas e expectativas dos usuários que dependem deles diariamente, não apenas do ponto de vista funcional mas, também, sob o aspecto da segurança. Podemos ter os seguintes controles associados com programas:

- **Controles Internos ao Programa:** trechos (ou a totalidade) do programa que buscam, unicamente, aumentar a segurança. Como exemplos podemos citas a implementação de controles de acesso, segregação de funções, trilhas de auditoria nas aplicações ou os mecanismos de integridade referencial existentes em um sistema gerenciador de banco de dados etc;
- **Controles do Sistema Operacional:** são limitações impostas pelo sistema operacional objetivando proteger os recursos de um determinado usuário dos demais usuários do sistema;
- **Controles no Desenvolvimento:** são os padrões de qualidade sobre os quais um programa é projetado, implementado, testado e mantido.

2.9.3. Controles por *Hardware*

Existem vários dispositivos de *hardware* que foram desenvolvidos para auxiliar na segurança computacional. Desde *hardware* especial que produz criptografia com maior segurança e desempenho até travas eletrônicas para portas. Mais recentes e com um grande aplicabilidade nas mais diversas áreas existem ainda as tecnologias de *tokens* físicos como *smartcards* [KPS95] e também um conjunto de dispositivos de biométrica que verificam a identidade de usuários através da íris dos olhos, da palma da mão ou da impressão digital. Também existem dispositivos de *hardware* que são usadas por desenvolvedores de *software* comercial para evitar a pirataria de seus produtos.

2.9.4. Políticas (normas)

Normalmente, os controles de sistemas computacionais são baseados em *software* ou *hardware* adicional, como os discutidos anteriormente. Entretanto, é possível incrementar a segurança através do estabelecimento de políticas [OLI94], que devem nortear o comportamento dos usuários. Controles simples como a mudança frequente de senhas e a utilização de senhas não triviais, proibição de instalação de programas e jogos, destruição de minutas (*drafts)* de documentos e relatórios impressos podem ser implementados praticamente sem custos e com resultados excelentes. O treinamento adequado dos usuários e o acompanhamento do cumprimento das normas estabelecidas são aspectos muito importantes e devem ser implementados juntamente com a definição das políticas.

2.9.5. Controles Físicos

Os controles físicos [MER98] são os mais fáceis de implementar, mais eficientes e mais baratos que os demais controles. Incluem travas nas portas, guardas para controlar o acesso de pessoas, cópias de segurança (*backups*) de *software* e dados relevantes, planos de contingência para desastres naturais etc. Entretanto, eles são comumente negligenciados, mesmo em instalações que implementam abordagens mais sofisticadas de outros tipos de controle. Não é incomum encontrarmos instalações que apesar de aplicarem inúmeras tecnologias avançadas de segurança permitem que um estranho (com farda e crachá da companhia telefônica, por exemplo), entre no CPD, tenha acesso ao *rack* de *modems* por algumas horas, normalmente desacompanhado, sem que a legitimidade da sua visita seja devidamente confirmada.

2.9.6. Legislação e Ética

Aspectos legais e éticos relacionados com a segurança computacional, apesar de importantes, são pouco desenvolvidos. A lei é lenta em sua evolução, ao contrário da tecnologia envolvendo computadores que tem se desenvolvido de forma alucinante. Embora proteção legal seja necessária e desejável, ela não é tão objetiva e eficaz nesta área como em outros tipos de crime, mais antigos e melhor entendidos e tratados. No aspecto da ética, a área de computação também é igualmente pouco desenvolvida, embora isto não signifique que profissionais da computação são, automaticamente, anti-éticos. Simplesmente, nem a sociedade em geral, nem a comunidade computacional em particular, definiram ou adotaram padrões formais de comportamento ético. Apesar do esforço de algumas organizações em ditar alguns códigos de ética para profissionais de computação, eles não são plenamente aceitos e muito menos aplicados em larga escala. Talvez falte ainda um consenso geral

sobre **quais** comportamentos são inadequados e **porque** são inadequados, como já ocorre em outras áreas do conhecimento (medicina, por exemplo).

2.9.7. Eficácia no Uso dos Controles

O simples fato dos controles estarem presentes e disponíveis não significa, automaticamente, que eles estão sendo bem usados. Existem alguns fatores que podem afetar a eficiência dos controles voltados para a segurança computacional.

O primeiro deles é o falta de cultura de segurança entre os usuários. Para se exigir que as pessoas cooperem com os requisitos de segurança é necessário que elas tenham consciência do problema e de como os controles de segurança devem ser usados em cada situação específica. Além da falta de sensibilização para as necessidades de segurança, também pesa o aspecto da dificuldade intrínseca de se automatizar processos, principalmente em situações onde um serviço ou funcionalidade está sendo implantada pela primeira vez, substituindo um processo manual ou uma tarefa anteriormente feita de forma centralizada no CPD. Neste casos, o usuário está mais concentrado em aprender o novo processo, ou seja, simplesmente realizá-lo corretamente. Realizá-lo com segurança torna-se, automaticamente, um objetivo em segundo plano.

Para minimizar o seu esforço, as pessoas tendem a restringir os seus atos ao que consideram o mínimo necessário para realizar uma determinada tarefa. Controles de segurança, infelizmente, não são vistos como fazendo parte deste "mínimo necessário". Obviamente, nenhum controle é eficaz se, embora disponível, não seja usado. Não adianta por uma trava eletrônica na porta no CPD, com um controle rígido de distribuição das credenciais para acesso, se um operador, para agilizar as várias entradas e saídas necessárias ao longo do dia, mantém a porta aberta. A aquisição e instalação de antivírus é inócua e o seu custo desperdiçado se um usuário desabilita no seu micro a checagem de vírus para diminuir o tempo gasto na inicialização. Da mesma forma, um sistema de *senhas* que permite forçar a mudança da senha periodicamente e também evita a utilização de senhas fáceis, disponível na maioria dos sistemas operacionais, não tem nenhuma validade se o administrador não ativa esta propriedade por causa do trabalho adicional que teria. Um caso clássico deste fenômeno ocorreu durante a Segunda Guerra Mundial, em que oficiais usavam códigos desatualizados para a cifragem de mensagens, em detrimento das ordens de usar códigos mais atuais, porque já tinham aprendido a usá-los e codificavam as mensagens mais rapidamente. Infelizmente, os inimigos já tinham quebrado alguns destes códigos e puderam visualizar o conteúdo das mensagens com facilidade.

Há a necessidade permanente de revisão dos controles de segurança. Poucos deles são permanentemente eficazes. Sempre que os especialistas em segurança desenvolvem uma nova técnica eficiente contra certos tipos de ataque, os atacantes duplicam os seus esforços na busca de uma maneira de anular o mecanismo de controle.

2.10. Planejamento das Necessidades de Segurança

Ter segurança prática é, realmente, mais uma questão de gerenciamento e administração do que um caso de se possuir maior ou menor habilidade técnica. Fundamentalmente, a segurança computacional é uma série de soluções técnicas para problemas não técnicos. Não importa a quantidade de dinheiro que se gaste em tecnologias de segurança, nunca será o suficiente para resolver totalmente o problema da segurança, ou seja, eliminar definitivamente os riscos. Pela quantidade de circunstâncias envolvidas – *bugs* de *software*, desastres naturais, acidentes, erros humanos ou atacantes bem motivados e equipados – qualquer sistema computacional pode, potencialmente, ser danificado ou comprometido de alguma maneira.

O objetivo do profissional de segurança é auxiliar a organização a decidir a quantidade de tempo e dinheiro que devem ser gastos com segurança na busca de um equilíbrio de **proteção adequada x custos limitados**. Antes da criação e implementação de políticas de segurança, o planejamento da segurança deve passar por dois momentos bem definidos: a **avaliação dos riscos envolvidos** e a **análise de custos e benefícios da segurança**. Discutiremos cada um deles a seguir.

2.10.1. Avaliação dos Riscos

Em segurança, antes de qualquer outra providência, é extremamente importante termos bem definidos **o quê** proteger, **porque** proteger e **do quê** proteger. Portanto, o primeiro passo para se aumentar a segurança computacional é tentar responder as seguintes questões básicas:

- **O que é necessário proteger?**
- **Contra o que a proteção é necessária?**
- **Quanto tempo, esforço e dinheiro é possível e vale a pena gastar para obter a proteção adequada?**

A **avaliação dos riscos** é um elemento muito importante no processo de segurança computacional, pois é fundamental conhecer os riscos potenciais para se realizar o

planejamento de que políticas e técnicas são necessárias para se reduzir tais riscos. São três os passos da análise de riscos [GS96]:

- Identificação dos componentes ou bens computacionais;
- Identificação das ameaças associadas;
- Cálculo dos riscos envolvidos.

Existem várias metodologias para se fazer tais levantamentos. Um dos métodos mais bem sucedidos baseia-se na realização de uma seqüência de *workshops* internos, envolvendo usuários, gerentes, técnicos e executivos da organização. Além de permitir a criação de uma lista mais completa de bens e ameaças, este processo participativo também auxilia na criação de uma cultura sobre a existência de riscos e da necessidade de segurança com todos os envolvidos. Outra forma é através da aplicação de técnicas e testes de aferição de vulnerabilidades [RID97] [MB98].

A avaliação dos riscos não é um processo pontual, realizado uma única vez. Pelo contrário, ela é um processo constante, a ser realizado periodicamente ou sempre que ocorra alguma mudança operacional ou estrutural relevante.

2.10.2. Análise de Custo/Benefício

Após o levantamento dos riscos presentes, é fundamental a atribuição de um custo para a ocorrência de cada risco detectado e também o custo de defesa contra ele. Este exercício é chamado de análise custo/benefício.

2.10.2.1. Custo da Perda

Calcular os custos envolvidos na perda de componentes computacionais não é uma tarefa trivial, pois a falta do serviço computacional pode parar ou comprometer o funcionamento de uma empresa. Uma abordagem simplista pode apenas levar em conta o custo de reparar ou substituir um item de *software* ou *hardware* particular. Entretanto, uma forma mais sofisticada de cálculo pode levar em consideração os custos implícitos ou perdas de se ter um serviço indisponível por um determinado período, o custo de retreinar recursos humanos, o custo dos procedimentos adicionais resultantes de uma perda, o custo da reputação da companhia, o custo de recuperar clientes perdidos etc. Desta forma, a complexidade do cálculo dos custos aumenta sempre que a acurácia desejada também é incrementada, principalmente ao se considerar também fatores mais subjetivos.

Normalmente, não é necessário calcular de forma exata o valor associado com cada risco possível, uma média aproximada já é suficiente. Por exemplo, a perda de um estoque de

papel para impressão e disquetes ou fitas virgens pode ser classificada como "abaixo de R$ 500,00" enquanto que a perda total por incêndio da sala principal do CPD pode ser classificada como "acima de R$ 1.000.000,00". Alguns itens também estão na categoria dos "irreparáveis ou insubstituíveis", que incluem a morte de uma pessoa-chave para a instituição ou a perda total do banco de dados de contas a receber, por exemplo.

Também é possível criar uma escala mais refinada para representar o custo da perda do que, simplesmente, "perdido ou não perdido". Um exemplo de classificação assim poderia ser a atribuição de valores para cada uma das ocorrências abaixo para cada um dos itens do sistema computacional:

- Não disponibilidade por um período curto (até 5 dias);
- Não disponibilidade por um período médio (até 2 semanas);
- Não disponibilidade por um periodo longo (mais de 2 semanas);
- Destruição ou perda permanente;
- Dano ou perda parcial acidental;
- Dano ou perda parcial intencional;
- Divulgação não autorizada interna (na própria organização);
- Divulgação não autorizada externa (concorrentes, imprensa etc);
- Recuperação ou substituição.

A partir da análise dos custos da perda, que ajuda a atribuir valores monetários para aspectos subjetivos, será possível identificar quais são os itens dos sistemas computacionais que são críticos por um critério universal: o valor financeiro da sua perda.

2.10.2.2. Custo da Prevenção

O custo da prevenção é outra forma de abordagem de um risco potencial e deve ser visto em conjunto com o custo da perda associado com o mesmo risco. Por exemplo, o custo da perda causada por uma falta de energia em uma instalação pode ser apenas a perda dos últimos dados digitados e o tempo de reinicialização do sistema (*reboot*). Por sua vez, o custo da prevenção pode ser a aquisição de um sistema de UPS (*Uninterruptable Power Supply*) para a empresa. A decisão de qual estratégia é mais adequada neste caso pode variar de empresa para empresa. Em algumas instalações, como agências bancárias, o custo da perda é inadmissível, pois implica na interrupção do atendimento para inúmeros clientes e também na possibilidade de perda de transações financeiras. Em escolas de

informática, por sua vez, em que os dados digitados pelos alunos não são muito relevantes, o custo da prevenção, com a instalação de *no-breaks* protegendo todos os laboratórios usados para as aulas é que pode ser impeditivo. Ao se calcular o custo da prevenção é importante também considerar a amortização do investimento pelo tempo de vida útil estimado do controle, quando possível.

Os custos envolvidos com a segurança computacional, tanto com a perda como com a prevenção, precisam ser balanceados de forma a se obter a melhor proteção com o menor desembolso monetário. A Figura 8, ilustra como os investimentos em segurança estão distribuídos no Brasil em 1998, conforme apurado na 4ª Pesquisa Nacional sobre Segurança da Informação.

2.11. Sumário

O *hardware*, o *software* e os dados são os principais componentes de um sistema computacional. Estas três peças em si e também o relacionamento existente entre eles, constituem a base das vulnerabilidades de um sistema computacional, que são fraquezas no sistema de segurança que podem ser exploradas em ataques intencionais ou não intencionais, possibilitados por ameaças, para causar perda ou dano aos sistemas computacionais. As quatro classes de ameaças que podem afetar sistemas computacionais são interrupção, interceptação, modificação e fabricação. *Hackers*, amadores e profissionais são os três tipos principais de criminosos digitais, ou seja, que realizam ataques intencionais. Face a tais riscos, é necessária a utilização de mecanismos de defesa e a segurança computacional consiste em garantir três características para os componentes básicos de um sistema: **confidencialidade**, **integridade** e **disponibilidade.** Três princípios a norteiam: **Princípio da Penetração mais Fácil**, **Princípio da Proteção Adequada** e **Princípio da Eficácia**. Dentre os mecanismos de controle e proteção disponíveis, destacam-se o uso de criptografia, controles por *software*, controles por *hardware*, controles físicos, políticas, normas, legislação e ética. Os controles precisam ser bem usados, buscando um equilíbrio de **proteção adequada x custos limitados**. A análise de riscos é um importante procedimento neste sentido, que consiste em avaliar o **quê** proteger, **porque** proteger e **do quê** proteger. Também é importante balancear tanto o custo da perda quanto o custo da prevenção, buscando uma relação custo/benefício favorável. No capítulo seguinte, abordaremos a questão da segurança voltada para aplicações distribuídas.

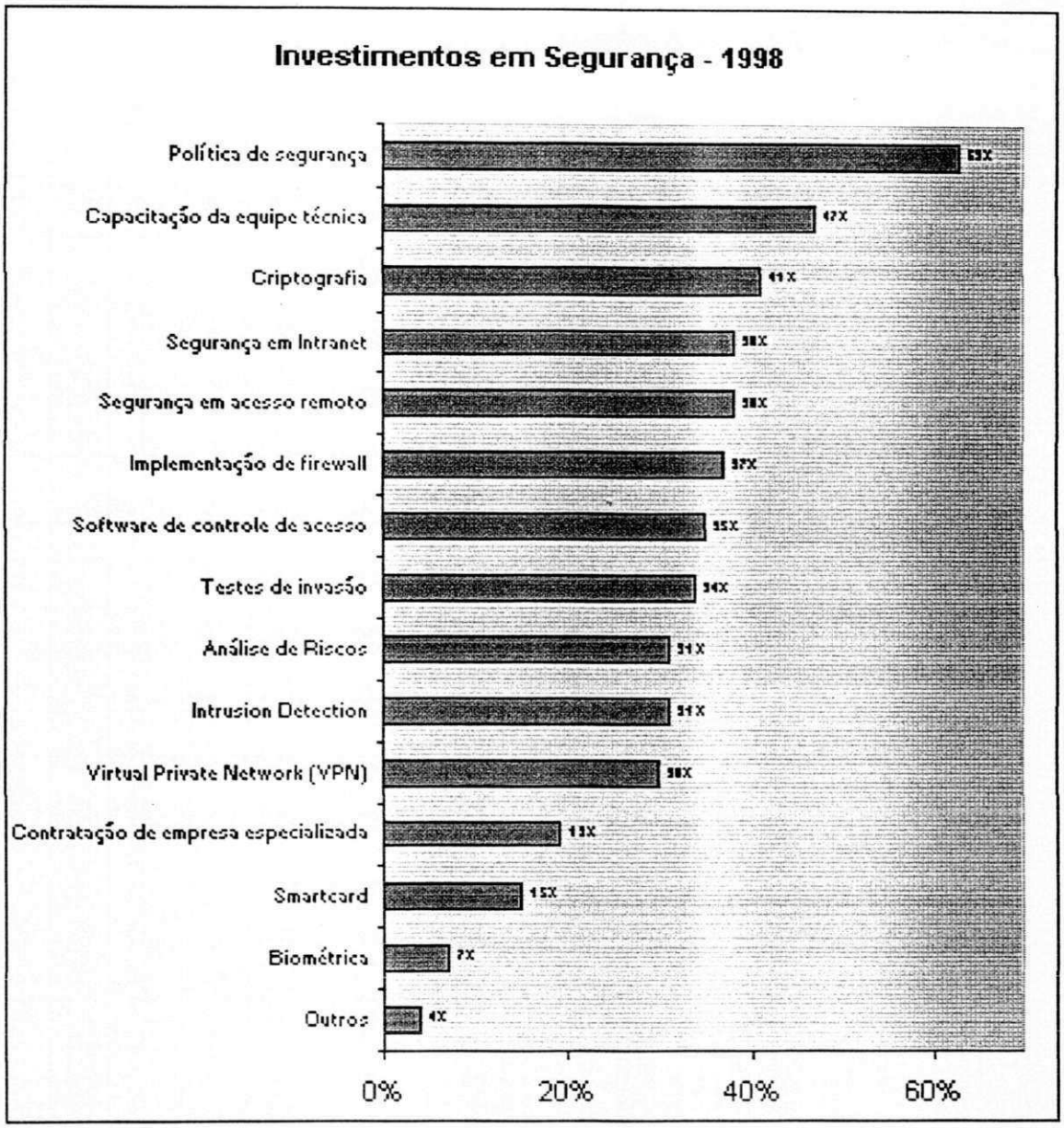

Figura 8: **Distribuição dos Investimentos em Segurança no Brasil**[14]

[14] Foi solicitado das empresas participantes da pesquisa que apontaram que algum investimento em segurança foi realizado em 1998 para que selecionassem várias respostas de uma lista múltipla de rubricas onde os investimentos em segurança foram feitos. Na consolidação, 69% das respostas

3. Segurança de Aplicações Distribuídas

A computação distribuída representa um conceito mais abrangente do que computação em rede [BIR96]. Uma rede de computadores é uma tecnologia de comunicação que suporta a troca de mensagens entre programas executando em diferentes locações. Ou seja, são "movedores de dados" (*data movers*), possibilitando o envio de dados de uma locação para outra. Podemos considerar, ainda de forma simplista, uma rede como uma coleção de computadores interconectados por *hardware* que suportam a troca de mensagens entre si. Sistemas distribuídos, por sua vez, são sistemas computacionais e aplicações que cooperam para a realização de ações coordenadas em múltiplas locações através de uma rede. Os sistemas distribuídos podem ser vistos também como um conjunto de programas sendo processados em um ou mais computadores e coordenando ações através da troca de mensagens. Por esta perspectiva, enquanto uma aplicação convencional (não distribuída) acessa dados remotamente através de uma rede, um sistema distribuído inclui múltiplas aplicações que se comunicam através da rede mas que realizam ações específicas, embora coordenadas, em várias locações.

Apesar do crescente desenvolvimento da tecnologia de conectividade nas últimas duas décadas, só recentemente sistemas computacionais distribuídos passaram a assumir um papel de maior destaque na indústria e na sociedade. Esta relativa "juventude" da área de sistemas distribuídos talvez justifique o fato de que tão poucos sistemas distribuídos são realmente confiáveis no sentido de oferecer características como tolerância a falhas automática, garantias de desempenho e tempo de resposta e mecanismos de segurança contra ameaças intencionais [BIR96]. O crescente uso de sistemas computacionais distribuídos em aplicações comerciais e para armazenar informações críticas (*sensitive data*) vem criando uma pressão significativa para se melhorar os mecanismos que incrementem a confiabilidade deste sistemas [NER98]. À medida que aumentam o número de aplicações críticas baseadas em sistemas distribuídos, onde a segurança e a estabilidade financeira de grandes organizações estão em jogo, cada vez mais os fornecedores de tecnologia estão sendo cobrados para demonstrar a confiabilidade de suas arquiteturas e soluções distribuídas [MGF97].

---

indicavam que eram gastos recursos no desenvolvimento de políticas de segurança, 47% indicaram algum tipo de recurso alocado para capacitação da equipe técnica e assim por diante.

O conceito de confiabilidade em aplicações distribuídas, como também em vários outros contextos, inclusive o do mundo real, dá margem a múltiplas interpretações, correspondendo cada uma a vários tipos de garantias diferentes, incluindo [BIR96]:

- **Especificação Correta** (*correct specification*): a garantia de que o sistema soluciona o problema desejado;
- **Implementação Correta** (*correct implementation*): a garantia de que o sistema corretamente implementa a especificação;
- **Tolerância a Falhas** (*fault tolerance*): a habilidade que um sistema tem de, mesmo na presença de falhas de componentes, continuar funcionando sem realizar ações incorretas;
- **Alta Disponibilidade** (*high availability*): a capacidade do sistema restaurar uma relativa operação, oferecendo serviços reduzidos, por um curto período de tempo, enquanto se reconfigura por causa da falha de algum componente;
- **Disponibilidade Contínua** (*continuous availability*): é um sistema altamente disponível com um tempo muito pequeno de recuperação, capaz de prover serviços ininterruptos aos seus usuários;
- **Recuperação** (*recoverability*): a capacidade de componentes que falharam se auto-iniciarem e se reintegrarem ao sistema, após a causa da falha ser sanada;
- **Consistência** (*consistency*): a habilidade do sistema de coordenar ações relacionadas através de múltiplos componentes, frequentemente em um contexto de concorrência e falhas. A característica de consistência, implicitamente, também se refere à habilidade de um sistema distribuído para emular um sistema não distribuído;
- **Desempenho Previsível** (*predictable performance*): a garantia de que o sistema obtém níveis desejados de desempenho (ex.: vazão de dados, latência, pedidos processados por segundo etc);
- **Pontualidade** (*timeliness*): em sistemas sujeitos a restrições de tempo real, a garantia de que as ações serão realizadas em determinados limites de tempo ou com um determinado grau de sincronização temporal entre os seus componentes;
- **Privacidade** (*privacy*): a habilidade do sistema de proteger a identidade e localização de seus usuários contra divulgação não autorizada;

- **Segurança** (*security*): a habilidade do sistema de proteger dados, serviços e recursos contra uso impróprio por usuários não autorizados.

De uma forma ou de outra, muitas destas garantias ou propriedades são relacionadas com questões de tolerância a falhas. Dentre os vários significados do termo falha em sistemas distribuídos [BIR96], uma categoria nos interessa em especial: são as falhas bizantinas[15], onde estão relacionadas as falhas associadas com a propriedade **segurança**. Como visto no capítulo anterior, os três principais elementos de sistema computacional sujeitos a ataques são *hardware*, *software* e dados, e as comunicações entre eles constituem a base das vulnerabilidades dos sistemas computacionais. A segurança computacional nada mais é do que garantir a confidencialidade, integridade e disponibilidade dos componentes de sistemas computacionais. Voltadas a oferecer tais características de segurança e vistas como um subconjunto das aplicações distribuídas confiáveis, as aplicações distribuídas seguras representam uma necessidade imediata do mercado.

Para atender tais espectativas e demandas, estão sendo, continuamente, desenvolvidas técnicas e mecanismos de controle voltados para sistemas distribuídos, tanto disponíveis em versões mantidas pela comunidade acadêmica quanto em produtos comerciais oferecidos pelos principais fornecedores de tecnologia. Independentemente do nome ou de abordagens específicas de cada um, estes controles podem ser agrupados em cinco categorias principais [BIR96]:

- Tecnologias de *firewall* e outros mecanismos de defesa perimetral (*perimeter defense*), que operam restringindo o acesso ou comunicação em limites ou fronteiras específicas. São populares mas limitados: uma vez que o intruso encontrou uma maneira de contornar (*bypass*) o *firewall* ou se conectar (*login*) no sistema, a proteção acaba.
- Mecanismos de controle de acesso, frequentemente baseados no modelo de identidade de usuário e de grupo do Unix para limitar o acesso a recursos compartilhados. Não se estendem a comunicações, o que é talvez o seu mais sério problema.

[15] As falhas bizantinas englobam uma grande categoria de comportamentos faltosos, incluindo corrupção de dados, quebra de protocolos e comportamentos adversos de programas que maliciosamente e ativamente buscam forçar um sistema a violar as suas propriedades de confiabilidade.

- **Mecanismos de proteção com manutenção de estado** (*statefull*), possuem conceitos fortes de estado de sessão e de canal. Realizam autenticação no momento em que a comunicação é estabelecida. Adotam a abordagem de que uma vez que o usuário é validado, a dificuldade de se infiltrar em uma sessão representa um obstáculo à intrusão.

- **Sistemas de autenticação**, usam algum esquema para autenticar o usuário que está executando cada aplicação. Sessões de comunicação são protegidas por alguma forma de chave, negociada por um agente confiável. Mensagens podem ser encriptadas ou assinadas digitalmente, resultando em garantias muito fortes. Apresentam custos altos de implementação: encriptação, assinaturas digitais, modificações não triviais em aplicações e necessidade de reautenticações frequentes.

- **Arquiteturas de Segurança Multi-Nível** (*MLS – Multilevel Distributed System Security Architecture*): são baseados em um padrão de segurança do governo americano, desenvolvido em meados dos anos 80. São extremamente robustos mas difíceis de implementar pois requerem muito esforço no desenvolvimento de aplicações. Talvez por isto sejam tão pouco usados.

Entretanto, mesmo com a aplicação dos mecanismos de segurança adequados, os sistemas, distribuídos ou não, ainda possuem pontos vulneráveis que, por serem exógenos, originados por fatores externos, são difíceis de controlar. Dentre os principais, podemos citar:

- **Usuários:** utilizam senhas triviais e são suscetíveis à engenharia social[16];

- **Administradores:** não implementam políticas de controle eficientes e cometem erros de configuração que comprometem a segurança;

- **Tomadores de Decisão:** frequentemente não têm consciência da necessidade de segurança;

- **Falhas de Projeto:** são comuns e ocorrem em vários níveis. Também são motivadas pela complexidade dos sistemas e por *bugs* de *software* e, além disso, durante o

[16] A engenharia social é um termo usado genericamente para definir um conjunto de técnicas e artifícios, ligadas ao relacionamento interpessoal, que permitem que sejam extraídas, de forma sutil, informações através de conversas espontâneas ou situações montadas artificialmente.

período de desenvolvimento dos sistemas, a segurança ou não é considerada ou é vista por último, como um aspecto secundário;

- **Vasta Informação Disponível:** a própria Internet ajuda a se ter acesso a ferramentas avançadas que popularizam as ações de *hackers* e *crackers*. Existem *websites* com informações detalhadas sobre falhas de segurança que permitem a utilização maciça de técnicas avançadas na violação da segurança das empresas (Ex: http://astalavista.box.sk);

- **Natureza Humana:** motivadas por curiosidade ou ganância, as pessoas tendem a cometer fraudes, principalmente no mundo virtual, considerado atraente e sedutor.

Outro fator preocupante é que o fraudador geralmente está próximo. Como citado anteriormente, estima-se que grande parte das violações de segurança são internas [PSI98], ou seja, alguém com acesso físico a computadores, com conhecimento de senhas e acesso aos dados da camada persistente mantidas tanto pelo servidor quanto pelos clientes.

3.1. Comércio Eletrônico e Segurança

Tais aspectos relacionados com a segurança computacional ficam ainda mais críticos quando associados a aplicações voltadas para comércio eletrônico, uma tendência mundial, e que possuem requisitos de segurança ainda mais específicos [FRA97]. A teia mundial ou *World Wide Web,* além de ser um ambiente propício ao desenvolvimento de aplicações distribuídas, também é um lugar extremamente generoso, pois as pessoas podem obter informações e serviços, não importando em que parte do mundo elas se encontrem, sem, necessariamente, precisar pagar por isso. Embora motivadas em parte por não ser uma tarefa trivial cobrar algo dos usuários, estas características atraíram e continuam a atrair os milhões de usuários que atualmente navegam na Internet. A convergência das empresas para a Grande Rede é quase que automática: **onde estão as pessoas, estão os negócios**.

Embora obter informação de graça seja muito bom, normalmente as pessoas estão dispostas a pagar por algo mais. As vantagens da utilização da Internet para que as pessoas comprem e as empresas vendam são enormes [BAU98]. Ganham as empresas que, além de estenderem globalmente o seu escopo de atuação, podem economizar com a produção e postagem de catálogos de produtos, evitar gastos com a manutenção de pontos de venda, realizar lançamentos e promoções *online* etc. Os consumidores, além de poderem procurar de forma confortável pelo produto ou serviço que desejam, lucram com a possibilidade de se beneficiar da livre concorrência de empresas do mundo todo em busca da sua preferência. Tudo isto realizado com um baixo custo operacional para ambas as

partes. Entretanto, realizar transações comerciais através da Internet requer um alto nível de segurança das aplicações distribuídas envolvidas [HIG97], que devem suportar os fundamentos básicos de transações eletrônicas seguras, incluindo [RUG97]:

- **Privacidade:** manter a confidencialidade de mensagens, transações, dados relevantes, chaves secretas etc. Esta característica baseia-se fortemente no uso de tecnologias de encriptação;
- **Autenticação:** validação da identidade das partes envolvidas na comunicação. Pode ser aplicada mutuamente pelo emissor e pelo receptor. Os principais recursos para se obter autenticação são a assinatura digital e a tecnologia de certificação digital [PAV98];
- **Integridade:** garantir que nada foi alterado nos dados da transação desde a sua fonte até o seu destino final. Os principais mecanismos utilizados são algoritmos de *secure hashing*, CRC polinomiais e tecnologias de autenticação de código, como *message digests* [KPS95];
- **Autoria Garantida:** também chamada de **Não Repudiação**, esta característica busca garantir que a transação seja reconhecida por ambas as partes. Baseia-se, principalmente, no uso combinado de assinaturas digitais.

Atualmente, o potencial da área de comércio eletrônico (*e-commerce*) tem despertado uma crescente demanda para a solução de vários problemas que impedem o desenvolvimento de aplicações desta categoria para serem usadas em larga escala [WAY97]. As pessoas tentam compatibilizar a possibilidade de fazer *megabytes* de informação cruzarem o mundo inteiro em frações de segundo e a limitação de não poder transportar um centavo sequer, nem para alguém a poucos quilômetros de distância.

Realizar transações eletrônicas com segurança é o grande desafio a ser vencido antes de termos um "mercado digital" pleno [BRO98]. Existem diversas tendências e tecnologias abordando o tema, algumas até já sendo utilizadas pioneiramente por algumas empresas. Dos mais simples aos mais complexos, algoritmos e técnicas como NetCash, NetCheque, CyberCash, CyberCoin, SET, CheckFree, CAFE, DigiCash, First Virtual, Open Market, S-HTTP, SSL e outros propõem-se a permitir o comércio digital e disputam o reconhecimento como um padrão nesta área [WAY97]. A principal corrente de pensamento, que engloba várias das técnicas disponíveis, defende uma analogia digital com mecanismos utilizados no mundo real, inclusive o conceito do dinheiro eletrônico [CAS97] e prega que quando "dinheiro digital" estiver realmente disponível, então transações verdadeiramente digitais serão possíveis. Podemos entender dinheiro verdadeiramente digital, ou universal, como aquele que pode: trafegar digitalmente, resistir a falsificações, oferecer privacidade,

funcionar *off-line*, ser transferido entre pessoas e ser decomposto em valores menores [OO92].

Entretanto, mesmo no mundo real, em que o dinheiro está totalmente integrado e, mais que isto, torna a vida moderna possível, os problemas de segurança são inúmeros. Apesar de todas as técnicas utilizadas na produção de papel moeda visando dificultar a sua reprodução, falsificações ainda representam um vasto nicho de criminalidade e as fraudes com cartão de crédito são uma enorme fonte de prejuízos para as administradoras. No mundo eletrônico espera-se que tais problemas de segurança aumentem pois, atualmente, copiar informação digital é muito fácil. Além disso, a impossibilidade de se exigir a presença física do cartão de crédito e a conferência da assinatura do cliente no momento de uma compra, usados como itens de segurança obrigatórios na forma tradicional de uso de cartões de crédito, tornam as transações digitais mais vulneráveis, pois o simples conhecimento do número do cartão de crédito de alguém já é o suficiente para se cometer uma fraude.

Segundo a 4ª Pesquisa Nacional sobre a Segurança da Informação, a popularização do comércio eletrônico ainda é vista com bastante desconfiança pelos usuários enquanto não forem resolvidas as questões de segurança. Apesar das altas projeções de crescimento no mercado mundial, apenas 23% das pessoas responderam que efetuaram alguma compra via Internet. No Brasil, começam a ser dados os primeiros passos para o *e-business*, com 31% das empresas realizando algum tipo de transação eletrônica via Internet (Figura 9). A principal aplicação continua sendo o uso de Home Bank/Internet Banking com 24% de uso (um aumento de mais de 20% em relação a 1997). Destacam-se ainda a transferência de arquivos, declaração de Imposto de Renda, compras via cartão de crédito, envio de cartas e telegramas, envio de informações para o Governo e serviço de divulgação de informações. No caso de empresas que realizam pagamentos via Internet, 66% usam o *protocolo Secure Sockets Layer - SSL* [HY95] [HIC95] para a segurança do processo, 32% utiliza criptografia proprietária e 2% informou que utiliza o protocolo SET [WAY97].

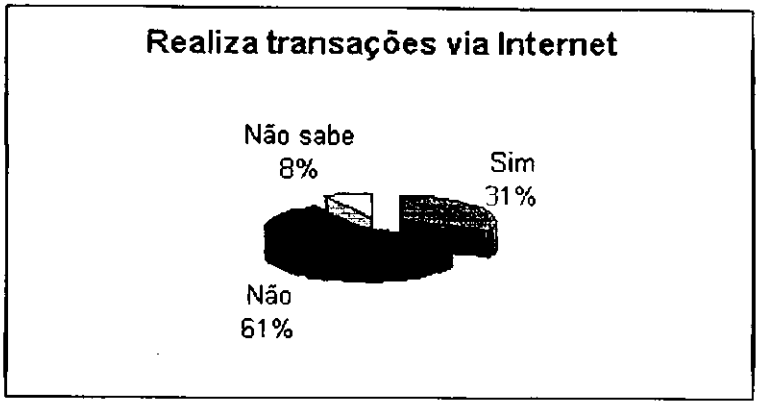

Figura 9: **Estatística de Empresas que Realizam Transações Via Internet no Brasil**

3.2. Segurança baseada em Senhas

As técnicas que buscam transações eletrônicas seguras em aplicações distribuídas como um todo, não apenas as de comércio eletrônico, têm, em sua grande maioria, as suas fundações baseadas em complexas teorias matemáticas [WAY97]. Em praticamente todos os casos, são usadas variações de criptografia e assinaturas digitais, onde existem senhas e outras informações consideradas privadas, que devem ficar fora do alcance de estranhos. A segurança do sistema é diretamente proporcional a segurança de tais senhas (ver Anexo **A).**

As senhas e outras informações que devem ser mantidas secretas, representam a base da maioria dos controles de segurança disponíveis, principalmente para a autenticação das partes envolvidas em uma transação. Por tal importância, elas representam um dos principais pontos de vulnerabilidade dos sistemas computacionais. Existem diversos problemas na distribuição e guarda de tais informações que devem ser solucionados, principalmente no lado da aplicação cliente.

O primeiro problema é a distribuição de senhas. Tanto os sistemas de autenticação baseados em criptografia simétrica (ou de chave secreta), como Kerberos [KPS95], quanto os assimétricos (ou de chave pública), como RSA [WAY97], fazem conceitualmente a mesma coisa, ou seja, distribuição de senhas, mas usando semânticas diferentes. A atividade principal (*fundamental security–enabling activity*) de um sistema de chave secreta é **emitir chaves** sob demanda e com baixa latência. Os sistemas de chave pública, por sua vez, têm como atividade fundamental **verificar a validade de chaves** em circulação (*as-yet-unrevoked status*), também sob demanda e com baixas latências. Este gerenciamento de chaves tem um custo, que em sistemas de chave secreta ocorre no momento da emissão da chave, enquanto que em um sistemas de chave pública, está relacionado com a revogação

da chave [GEE98]. Por tais diferenças semânticas, os sistemas criptográficos simétricos e assimétricos possuem diferentes campos de atuação. Por serem mais rápidos e oferecerem revogação de chaves sem custos, os sistemas simétricos são a escolha natural para uso dentro das organizações. Sistemas de chave pública são mais lentos, mas permitem assinatura digital e autoria garantida, o que os habilitam para uso entre organizações. Ambos, entretanto, são complexos e de implantação difícil, o que dificulta o seu uso em larga escala, principalmente em aplicações de menor porte. Existem também algumas críticas ou ressalvas com relação a sua adoção [BIR96]. O Kerberos, por exemplo, é bastante criticado por não usar um algoritmo assimétrico como o RSA e sim o algoritmo simétrico DES [KPS95]. Além disso, o servidor onde o Kerberos executa deve ser física e lógicamente seguro. Do lado do RSA, existe o problema da patente[17] e de restrições de exportação das versões mais poderosas por parte do governo dos Estados Unidos, além de exigir um grande esforço na montagem de uma PKI (*public key infraestruture*) [18] [HIL97] [SDT99].

Com relação à guarda de senhas e outras informações secretas, também há diversos problemas. O principal deles é manter as senhas longe do alcance de estranhos. Quando falamos de senhas de um sistema, com poucos caracteres, até que é possível guardá-las na memória, o que, pelo nível de tecnologia atual, talvez seja o local mais inacessível e seguro para isto. Entretanto, com o advento de chaves privadas isto ficou impraticável, já que tratam-se, normalmente de longas cadeias de caracteres, quase que aleatórias, sendo impossível para um humano memorizá-las. Estas informações normalmente são armazenadas em arquivos, como também os certificados digitais que as validam. A proteção necessária atualmente não é mais preservar a senha como dado ou informação na memória de uma pessoa, mas, unicamente, garantir a inviolabilidade do repositório escolhido para armazená-la. A captura de um arquivo destes permite a aplicação de criptoanálises (ver Anexo A) e a eventual descoberta da senha.

3.3. Sumário

Os sistemas distribuídos são sistemas computacionais e aplicações que cooperam para a realização de ações coordenadas em múltiplas locações através de uma rede. O uso

---

[17] Em resposta a tais questões, foram desenvolvidos algoritmos similares ao RSA sem essa restrição.

[18] O uso de criptografia de chave pública requer a existência de uma estrutura com serviços essenciais para gerenciamento de certificados digitais e chaves de encriptação para pessoas, programas e sistemas.

crescente de sistemas computacionais distribuídos em aplicações comerciais e para armazenar informações críticas (*sensitive data*) vem exigindo a melhoria dos mecanismos voltados para a segurança destes sistemas. Tais aspectos relacionados com a segurança computacional ficam ainda mais críticos quando associados a aplicações voltadas para comércio eletrônico, uma tendência mundial, e que possuem requisitos de segurança ainda mais específicos. Mecanismos de defesa perimetral, de controle de acesso, de proteção *statefull*, de autenticação e de segurança multi-nível são as principais categorias de controles para sistemas distribuídos. Em praticamente todos os casos, são usadas variações de criptografia e assinaturas digitais, onde existem senhas e outras informações consideradas privadas, que devem ficar fora do alcance de estranhos. A segurança do sistema é diretamente proporcional à segurança de tais senhas. No próximo capítulo, vamos discutir o modelo de segurança fim-a-fim, como uma alternativa para aumentar a segurança de aplicações distribuídas.

4. Modelo de Segurança Fim-a-Fim

Aplicações distribuídas estão sempre expostas a ataques e/ou falhas, estejam ou não disponibilizadas através da Internet. Fraudes e erros humanos são comuns de ocorrer e extremamente difíceis de detectar. Os sistemas, frequentemente, ficam expostos a sua ocorrência, pois normalmente eventos assim não são considerados quando os controles de segurança são implementados nas empresas. Além disso, mesmo a segurança tradicionalmente praticada possui falhas e brechas fáceis de serem vencidas e publicamente conhecidas [HUR97], que para serem exploradas necessitam apenas de alguém com um bom motivo e um pouco de determinação. Uma causa principal para isto é que as senhas usadas para garantir a autenticação de um usuário, mesmo as não triviais, normalmente são estáticas. Se o fraudador descobre a chave adequada, pode se passar integralmente pelo dono original até ser detectado, o que pode significar muito tempo ou grandes prejuízos. Até mesmo *tokens* físicos como os *smart cards*, também usados para autenticação e considerados extremamente seguros, já foram fraudados[19]. Em fraude, tudo se resume a que os resultados justifiquem os riscos e o esforço para burlar a segurança, ou seja, as necessidades de incremento da segurança de uma aplicação aumentam proporcionalmente ao benefício que as pessoas podem tirar de uma eventual fraude.

4.1. Modelo de Segurança Tradicional

Normalmente, a segurança dos sistemas, especialmente aqueles com enfoque distribuído, está concentrada em dois pontos principais: i) **segurança perimetral ou de rede**, que envolve *firewalls*, criptografia, assinaturas digitais e outros mecanismos visando estabelecer conexões seguras, privativas e com autenticação das partes envolvidas [KPS95]; e ii) **segurança interna**, que engloba o uso de senhas, a proteção contra sinistros, a proteção dos dados nas camadas de persistência, o controle físico e de acesso etc [HUN94]. A Figura 10 ilustra o modelo de segurança tradicionalmente adotado.

[19] Um caso destes foi reportado no Japão em maio de 1996, com perdas de mais de 500 milhões de dólares [WSJ96a] [WSJ96b]

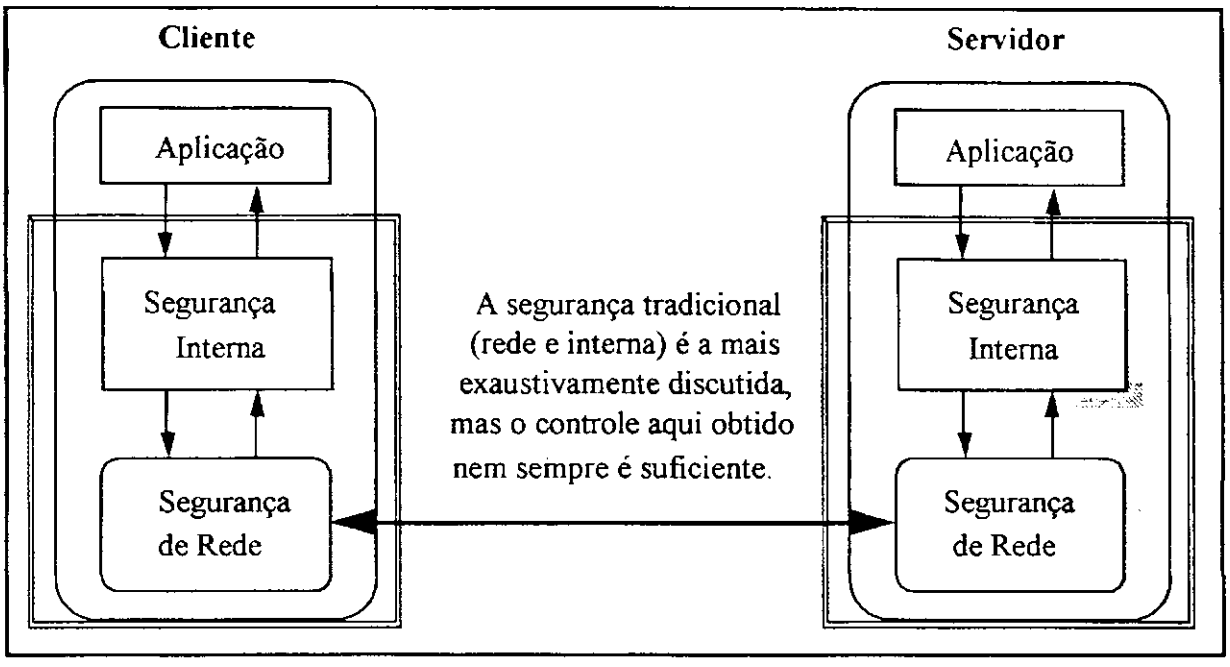

Figura 10: **Modelo de Segurança Tradicional**

A prática da segurança tradicional, mesmo considerando-se a utilização das mais variadas técnicas disponíveis em diversas combinações, nem sempre é suficiente pois não atinge todos os níveis de uma forma integrada. Normalmente, quando se discute segurança, há uma grande preocupação com os comprimentos de chave e outros aspectos técnicos da criptografia. Embora sejam questões relevantes, estas nem sempre representam o perigo real. Alterações indesejáveis em um arquivo de banco de dados, podem causar os mesmos danos, não importando se ocorreram de forma acidental ou de forma intencional. É importante, portanto, adotar mecanismos de segurança que possam cobrir tanto os riscos internos quanto os externos.

Neste sentido, um aspecto a ressaltar é que muita importância é dada à segurança das mensagens de transações - o que é compreensível e esperado em ambientes distribuídos, entretanto a segurança da camada de armazenamento de dados tem recebido uma menor atenção. Embora os SGBD´s também estejam presentes em muitas aplicações distribuídas, o controle de acesso aos dados é normalmente limitado à proteção por senha simples. Este relativo descaso na proteção dos dados na camada persistente se constitui em um maior risco, pois embora as mensagens tenham vida curta, as informações em bancos de dados são normalmente armazenadas por períodos muito mais longos e o tempo que um *hacker* tem para trabalhar é um fator determinante no grau de vulnerabilidade das informações. Apesar de existirem versões de SGBD´s seguros [BIR96], aplicados principalmente na área militar, poucos ambientes comerciais se utilizam de tal proteção ou de outras baseadas em

criptografia. Desta forma, assumindo que foram adotadas todas as medidas para dar segurança às mensagens, a maneira mais fácil de danificar ou fraudar dados, neste contexto, seria através do acesso aos bancos de dados, relativamente desprotegidos [WHI97].

Para exemplificar isto, vamos tentar criar um cenário hipotético, de uma instalação com extrema segurança física, com uma rede bem administrada e bem protegida, com *firewalls* restringindo corretamente o acesso de serviços apenas para os usuários autorizados. As aplicações, extremamente bem especificadas e implementadas, não possuem *bugs* e usam o modelo cliente/servidor com um bom gerenciador de banco de dados (SGBD) [DAT91] na retaguarda e uma linguagem visual moderna na camada de aplicação. Os usuários por sua vez, escolhem bem as suas senhas, seguem à risca as políticas internas e estão bem instruídos para se protegerem de acesso baseados em engenharia social. Vamos supor que neste "mundo perfeito", existe um funcionário, apenas um, que está insatisfeito e suficientemente motivado para cometer uma fraude ou causar dano à empresa. Mesmo não sendo um usuário privilegiado e não possuindo grandes privilégios de acesso ao sistema, ainda assim é possível que ele cometa danos graves nesta nossa instalação hipoteticamente perfeita.

A principal falha que pode ser explorada aqui é que, normalmente, a mesma senha que permite ao usuário insatisfeito acessar a aplicação também permite que ele se conecte ao banco de dados diretamente, através de uma ferramenta interativa como o **sqlplus**, no caso do SGBD Oracle, ou como o **dbexplorer**, no caso da linguagem Delphi. Como ambas vão se utilizar do mesmo serviço de rede para acessar o SGBD, ativado pelo mesmo usuário, usando a mesma estação, o *firewall* não detectará nenhuma anormalidade. Então temos o problema, pois quando o usuário usa a aplicação está sujeito ao crivo da segregação de funções, tendo, por exemplo permissão apenas de consultar alguns dados e emitir alguns relatórios mas, quando acessa diretamente o SGBD, a única restrição que ele terá pela frente serão as impostas no próprio banco de dados e voltadas para a manutenção da integridade referencial das tabelas.

Embora seja possível minimizar esta vulnerabilidade com uma utilização mais precisa dos mecanismos de segurança do próprio banco de dados, ou mesmo através da criação de um sub-sistema de segurança na própria aplicação, usando senhas intermediárias para não fornecer a senha real de acesso ao SGBD para o usuário, não é o que ocorre na prática. O desenvolvedor de aplicações está bastante concentrado em cumprir prazos e atender às demandas por funcionalidades dos sistemas e considera que a segurança da instalação é de responsabilidade do técnico de suporte. O técnico de suporte, por sua vez, possui pouco

domínio sobre a aplicação em si, aplicando, na maioria das vezes, soluções genéricas que atuam apenas até a periferia das aplicações. Esta falta de integração se traduz em componentes pouco acoplados, onde os mecanismos de segurança usados nas camadas anteriores não garante totalmente a legitimidade das transações realizadas na aplicação. Além disso, a prevenção e detecção de fraudes ou falhas fica extremamente mais difícil, principalmente porque a falsa autenticação de um intruso como um usuário legítimo não deixa marcas muito claras.

Atualmente, temos disponíveis a maioria dos componentes técnicos necessários para prover de segurança os sistemas distribuídos. O que ocorre é que não é fácil combiná-los em um sistema de segurança eficaz. Várias camadas de segurança, atuando de forma incremental, além da complexidade de implementação, nem sempre se constituem em um elemento verdadeiramente integrado. Se houver falha em alguma parte, o sistema pode ficar vulnerável. Portanto, mecanismos que auxiliem na detecção destas falhas podem minimizar potenciais danos e prejuízos. A seguir, discutiremos um modelo de segurança que também envolve a camada de aplicação na proteção da integridade das transações.

4.2. Adicionando mais um Nível de Segurança: Modelo Fim-a-Fim

A segurança de transações pode, e deve, ser vista como um processo fim-a-fim, ou seja, incluindo também os aplicativos finais que implementam as regras de negócio em ambos os lados, cliente e servidor, e envolvendo inclusive os mecanismos de proteção aos dados armazenados na camada de persistência, usando ou não SGBD´s. Seguindo esta filosofia, a maioria das técnicas de comércio eletrônico são materializadas por aplicações completas, integradas, voltadas fortemente para a segurança. Embora as necessidades de aplicações mais convencionais não sejam tão críticas e apesar da maciça utilização de tecnologias de segurança em camadas intermediárias, é fundamental que as partes finais envolvidas, também mantenham um controle sobre as transações e modificações realizadas nos respectivos contextos. Assim, respeitadas as devidas competências de atuação, cada camada deve implementar mecanismos de controle de tal forma que os fundamentos de segurança de transações sejam sempre observados [BIR96].

Por esta abordagem, além da usual segregação de funções, a partir do controle de **que** operações podem ser realizadas **por quem**, e das trilhas de auditoria que permitem a criação de domínios de responsabilidade para usuários e o registro das operações realizadas para posterior rastreamento, é interessante implementar, na camada de aplicação, um controle adicional de segurança servindo como barreira final contra intrusos ou, em última instância, como mecanismo de detecção da quebra da segurança. Este

controle fim-a-fim possibilita que, permanentemente e dinamicamente, as aplicações cliente e servidor validem mutuamente o contexto comum. O modelo de segurança aqui proposto é ilustrado na Figura 11.

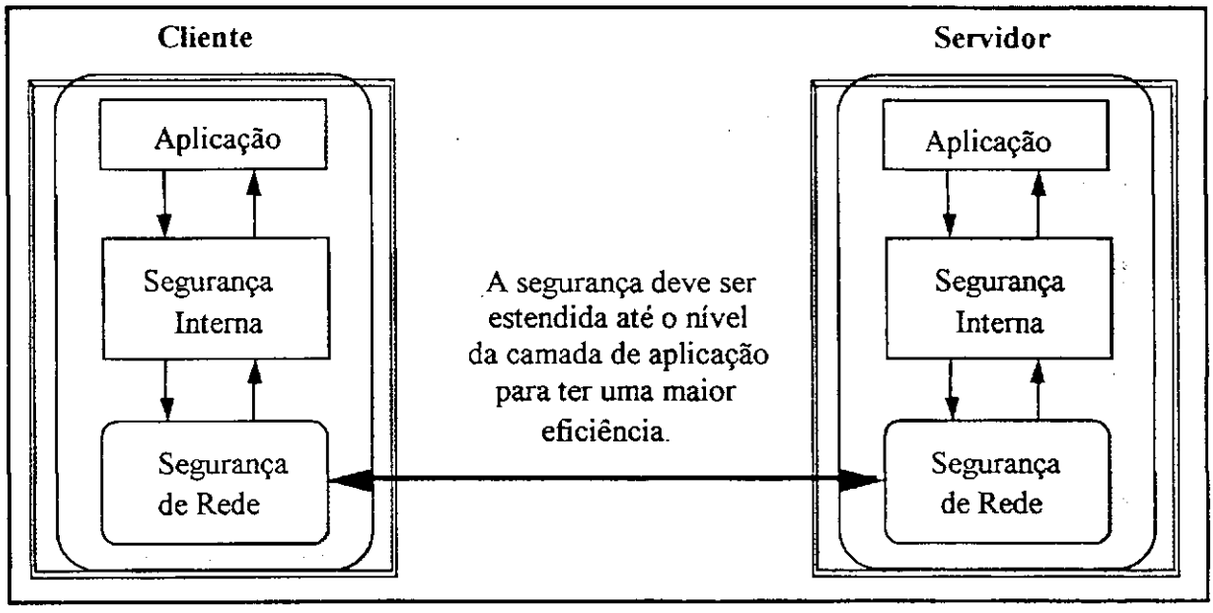

Figura 11: **Modelo de Segurança Fim-a-Fim**

Os objetivos principais da implementação de mecanismos adicionais de segurança na camada fim-a-fim seriam:

i) Permitir a detecção de ataques bem sucedidos, principalmente os relacionadas com quebra de senhas, que conseguiram atravessar as camadas de segurança de rede e segurança interna (se existentes) e alterar dados; e

ii) Permitir a detecção de erros humanos e falhas de *hardware* ou *software* que gerem inconsistência dos dados, incrementando assim os mecanismos de tolerância a falhas do sistema.

Para ser viável e eficaz, o controle de segurança fim-a-fim deve atender a algumas premissas básicas:

i) O controle adicional de segurança deve ser simples, possibilitando a sua implementação até em sistemas existentes. Não se justifica um controle de segurança que aumente substancialmente o nível de complexidade da aplicação. Obviamente, existem algumas aplicações que exigem um nível

maior de proteção. Estas podem bancar o custo do aumento de complexidade;

ii) O desempenho final do sistema não deve ser comprometido. O tempo de processamento, os custos com armazenamento e recuperação de dados e o tráfego na rede já se constituem em fatores críticos e acenos com perda de performance podem inibir a sua adoção; e

iii) O controle deve ser independente de qualquer tecnologia, inclusive das utilizadas nas camadas intermediárias de segurança, se existentes. É desejável a possibilidade de adaptação a qualquer linguagem, SGBD, algoritmos de criptografia, protocolo de rede etc.

Outra vantagem da utilização do controle na camada fim-a-fim é que os mecanismos de autenticação podem ser incrementados para permitir a autenticação de dois fatores (*two factor authentication* ou *strong authentication*) [LOB99]: alguma coisa que você **sabe** e alguma coisa que você **tem**, diminuindo assim a dependência da segurança da aplicação em relação à segurança das senhas utilizadas. Embora este tipo de autenticação seja mais comum com o uso de *tokens* físicos, é possível construir uma espécie de *token* lógico baseado no conhecimento do relacionamento anterior entre as partes. No próximo capítulo, veremos como construir este tipo de autenticação.

4.3. Sumário

A segurança dos sistemas, especialmente aqueles com enfoque distribuído, está normalmente concentrada em dois pontos principais: **segurança perimetral** e **segurança interna**. Esta segurança tradicionalmente praticada, nem sempre é suficiente pois não atinge todos os níveis de uma forma integrada. Por isto, a segurança de transações pode, e deve, ser vista como um processo fim-a-fim, ou seja, incluindo também os aplicativos finais que implementam as regras de negócio em ambos os lados, cliente e servidor. Os objetivos principais da implementação de mecanismos adicionais de segurança na camada fim-a-fim seriam a detecção de ataques bem sucedidos e a detecção de erros humanos e/ou falhas. Para ser viável, o controle fim-a-fim deve ser simples de implementar, usar baixo processamento e ser independente de tecnologia. Veremos no próximo capítulo uma proposta de algoritmo que permite criar um mecanismo de controle na camada de aplicação a partir de ferramentas simples e com um baixo custo computacional.

5. Encadeamento de Transações: Um Controle Fim-a-Fim

Quais os ganhos de se despender esforço para prover as aplicações, normalmente centradas apenas na implantação das regras de negócio, de mecanismos adicionais de segurança? Qual o tamanho de tal esforço? Como conciliar tal esforço com os objetivos principais da aplicação?

Para tentar obter uma resposta para tais questões vamos apresentar uma proposta de controle fim-a-fim básico, mas que sirva como modelo de segurança adicional para aplicações distribuídas e que atenda às premissas estabelecidas no capítulo anterior. Baseado em um conceito simples, o de criar uma contextualização de transações entre as aplicações cliente e servidor, discutiremos a seguir o **encadeamento de transações**.

5.1. O que é o Encadeamento de Transações?

Em aplicações comerciais e financeiras tradicionais como a movimentação de uma conta corrente ou a realização de compras com um cartão de crédito, cabe tanto à instituição quanto ao correntista realizarem constantemente uma verificação do andamento do "relacionamento" entre ambos.

Assim, se o cliente extrapola os seus limites, cabe à instituição detectar tal situação e tomar providências no sentido de resgatar uma posição de equilíbrio, quer seja inibindo novos créditos e convidando o cliente para fazer um depósito ou, dependendo da situação, aumentando o limite de crédito do cliente. Da mesma forma, cabe ao cliente administrar a chegada dos créditos e débitos em sua conta, detectando situações anormais e buscando os devidos esclarecimentos ou soluções junto ao banco. Com esta parceria e eterna vigilância mútua, fraudes e exceções são mais facilmente detectadas e resolvidas.

Vamos, agora, imaginar um diálogo hipotético em que alguém, ao encontrar outra pessoa, perguntasse: *Olá, lembra de mim? Eu sou o fulano, amigo de sicrano. Eu quero comprar um quilo de tomates.* Na próxima vez, o diálogo seria assim: *Olá, lembra de mim? Eu sou o fulano, amigo de sicrano, que lhe comprou um quilo de tomates. Agora eu quero comprar dois quilos de cebolas.* Na terceira vez: *Olá, lembra de mim? Eu sou o fulano, amigo de sicrano, que lhe comprei um quilo de tomates e depois dois quilos de cebolas. Desta vez preciso de uma dúzia de laranjas.* E assim por diante. Embora de uso improvável no dia-a-dia, esta forma de se fazer lembrar é uma maneira eficiente de autenticação. Com a mesma filosofia deste exemplo real, é possível criar um contexto entre as partes cliente e servidor de aplicações distribuídas dividindo entre cada uma das partes o controle e detecção de

situações de anormalidade. Com a manutenção por parte da aplicação cliente de uma espécie de resumo, com um mínimo de informações sobre as transações que trocou com a aplicação servidor desde um dado momento, e a checagem por parte do servidor se tal informação mantida pelo cliente confere com os dados mantidos na sua base persistente, é possível criar um encadeamento entre as transações efetuadas por ambos.

Para um melhor entendimento, vamos usar a mesma analogia de um relacionamento entre um banco e os seus clientes: o encadeamento de transações funcionaria como se a aplicação cliente mantivesse o controle do canhoto dos cheques emitidos e, a cada novo cheque emitido, incluísse informações sobre o cheque anterior. O banco, no caso o processo servidor, além de conferir a assinatura, saldo, limite do cheque especial, também faria a checagem dos dados sobre o cheque anterior para ver se quem emitiu o novo cheque tinha conhecimento do anterior e assim por diante[20]. Outro exemplo é o relacionamento entre uma loja eletrônica e os seus clientes: o encadeamento de transações funcionaria como se a aplicação cliente mantivesse o controle das compras realizadas e, a cada nova compra, incluísse informações sobre a compra anterior. O processo servidor, no caso a loja eletrônica, além de conferir os dados do cliente e da compra atual, também faria a checagem dos dados sobre a compra anterior para ver se quem está tentando realizar a nova transação tinha conhecimento da anterior e assim por diante.

Encadeando uma transação com a sua antecessora, criamos então um contexto em que uma transação só será válida se referenciar como anterior a última transação registrada pelo servidor. Ou seja, mesmo uma transação legítima só será considerada como tal se chegar na seqüência em que foi gerada e sendo esperada como a próxima pelo servidor. Assim, a autenticação entre as partes passa a ser realizada de forma permanente, desde o momento em que a transação é composta pelo módulo cliente até a sua efetivação pelo módulo servidor, em uma espécie de desafio mútuo[21].

---

[20] Embora isto não seja possível a partir da compensação tradicional de cheques, que é totalmente *offline*, baseada em um modelo que usa documentos físicos e onde a ordem de chegada dos cheques não é *FIFO – First In First Out*, tal mecanismo é plenamente viável no novo modelo bancário, que é eletrônico e *online*, baseando-se no auto-atendimento e no uso do cartão magnético para saques e outras operações. É neste novo modelo, também usado no comércio eletrônico, que estamos interessados aqui.

[21] Também é possível para o cliente validar o servidor. Isto é tratado mais adiante, quando discutimos o uso do *hash* derivado.

qualquer momento. Desta forma, simples processos-lote (*batch*) podem ser usados para varrer a base de dados em busca de alterações que deixaram tais valores inconsistentes. Observe que chamamos as ocorrências anormais em que novas transações são rejeitadas ou transações já realizadas não são validadas de **exceções** e não **fraudes**, pois erros humanos involuntários e falhas de *hardware* ou *software* também são detectados.

- **Dados Persistentes no Cliente**: Um mecanismo de registro do contexto do relacionamento entre o cliente e o servidor deve ser disponibilizado para a aplicação cliente. Qualquer processo que se preste para este fim poder ser utilizado, desde o registro simples a partir da gravação de um arquivo no disco rígido de um microcomputador até a utilização de *hardware* específico. Tudo depende dos riscos envolvidos, da criticidade da aplicação e do nível de segurança desejado.

5.3. Passos do Encadeamento de Transações

Para ilustrar melhor, vejamos como, utilizando-se dos elementos discutidos anteriormente, as transações podem ser encadeadas para a construção do contexto entre o cliente e o servidor.

- **Primeiro Momento: Cliente monta a Transação Contextualizada**

  A composição de uma transação **encadeada** ou **contextualizada** baseia-se no uso de uma função de *secure hashing*[22], que associa um código unívoco aos dados da transação concatenados com o código unívoco da transação anterior. O número assim obtido é recalculado pelo servidor e comparado com o que foi enviado juntamente com a transação pelo cliente. Ao compor uma nova transação, o módulo cliente da aplicação deve utilizar a função de *hashing* adotada sobre os dados específicos da transação e também sobre o valor de *hash* da última transação realizada. Obtém-se então o valor de *hash* para a nova transação, de forma encadeada com a transação anterior que, por sua vez, já estava encadeada com a sua antecessora e assim por diante. A transação assim composta deve ser enviada para o módulo servidor obedecendo aos critérios de segurança tradicional implementados. O valor de *hash* da última transação deve ser armazenado em uma camada persistente, sendo mantido de sessão para sessão. Se a aplicação cliente usa chaves para se autenticar junto ao servidor, o mesmo mecanismo

[22] Vamos usar também apenas a expressão *hashing* para indicar a aplicação da função de *secure hashing*.

utilizado para guardar tais chaves também pode ser usado para armazenar o contexto. Mais adiante, na seção 6.5, discutiremos o tratamento de exceções, inclusive as relacionadas com o vazamento do valor do hash e a sua eventual captura/utilização por um intruso.

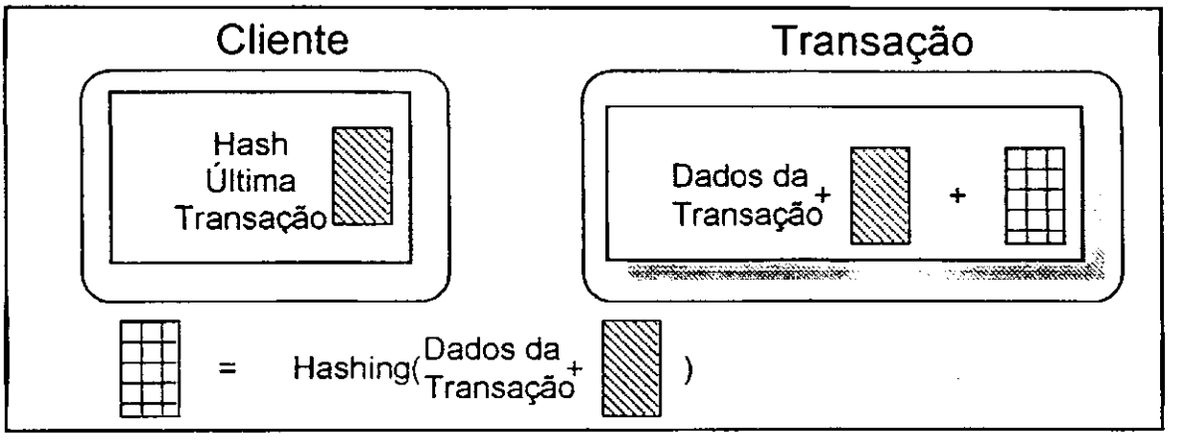

Figura 12: Montagem de uma Transação Contextualizada

- **Segundo Momento: Servidor valida a Transação Contextualizada**

  Ao receber uma nova transação de um determinado cliente, o módulo servidor deve, inicialmente, aplicar as técnicas de autenticação e validação previstas na segurança tradicional. Vencida esta etapa, o próximo passo é verificar a integridade da transação, conferindo se o *hash* enviado pelo cliente na transação confere com os dados contidos na própria transação, o *hash* anterior inclusive. Para isto, a aplicação servidor repete o mesmo processo utilizado pela aplicação cliente para o cálculo do *hash* atual. O último passo do processo de autenticação é verificar se o *hash* anterior incluído na nova transação bate com o *hash* da transação anterior. A aplicação servidor faz isto simplesmente consultando a sua base de dados persistente, através de uma comparação simples.

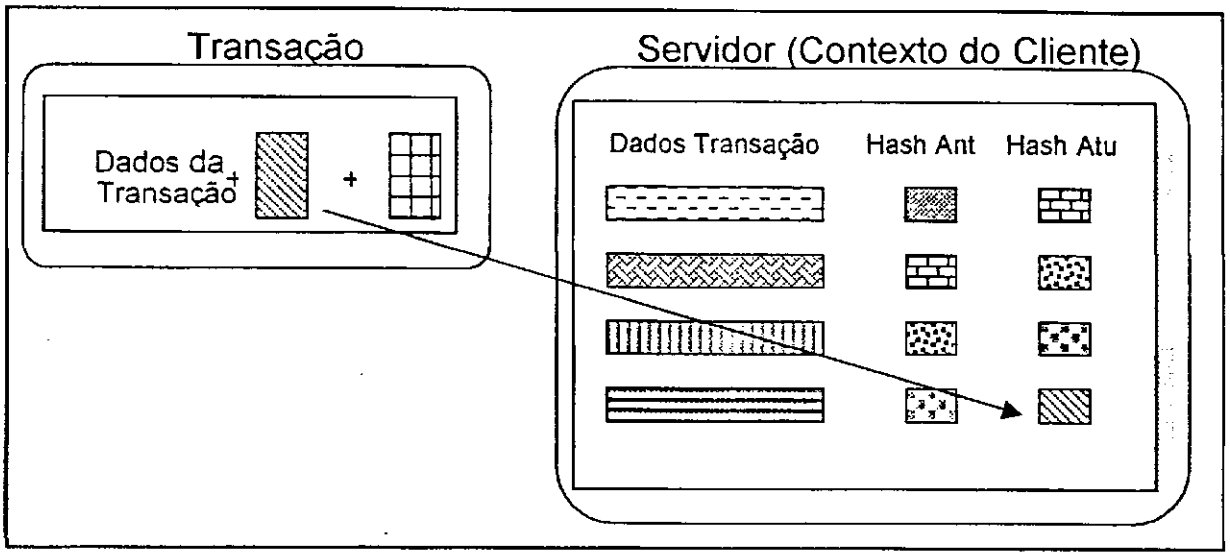

Figura 13: **Validação da Transação Contextualizada**

- **Terceiro Momento: Servidor registra a Transação**

  Em caso de sucesso, a transação pode ser efetuada, verificando-se os procedimentos normais de um ambiente transacional [BIR96]. Junto com os dados normais da transação, o servidor deve armazenar tanto o *hash* anterior informado quanto o *hash* atual.

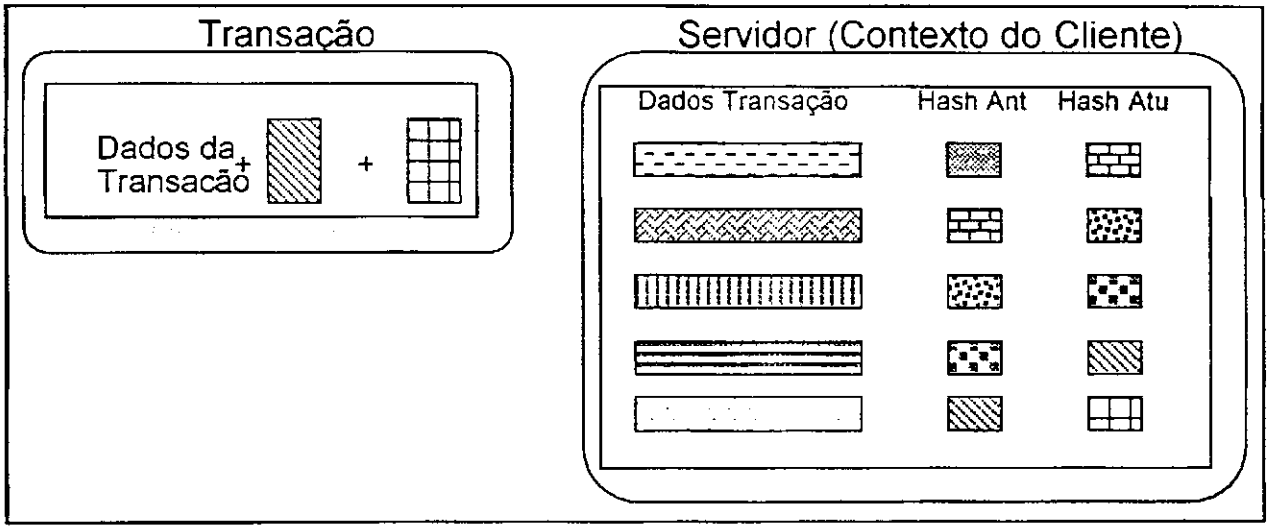

Figura 14: **Registro da Transação Contextualizada no Servidor**

- **Quarto Momento: O Cliente atualiza dados sobre o Contexto**

  O último passo realizado pelo cliente é memorizar, de forma persistente, o *hash* da última transação realizada com o servidor para poder repetir o processo adequadamente. Como citado anteriormente, é necessário proteger o valor do *hash* da última transação para evitar que o mesmo caia em mãos erradas. Uma das formas de

fazer isto é através da aplicação de criptografia. Em casos mais extremos, há sempre a alternativa de se armazenar o *hash* em uma mídia física mais segura, por exemplo em um *smartcard*.

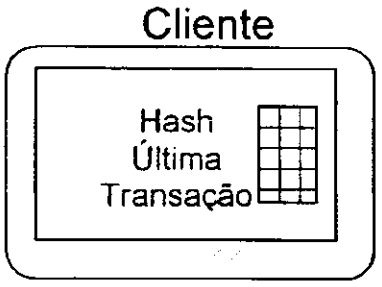

Figura 15: **Registro do Contexto pelo Cliente**

Na **Figura 16**, que contém algumas mensagens possivelmente enviadas por uma aplicação cliente hipotética, podemos observar um exemplo de encadeamento de transações para o caso da aplicação bancária discutido anteriormente. Foi usado o algoritmo MD5 para o cálculo do *secure hash* e considerado um estado inicial vazio do *hash* anterior para a primeira transação.

DATA	NUMERO	HIST.	DOCTO	CONTA	TIPO	VALOR	HASH ANTERIOR	HASH ATUAL
19981201	0000001	001	000001	1234567890	D	00000100000	ϕ	a1911b03913062ee42c8a907a8cba145
19981203	0000023	002	000002	1234567890	C	00000300000	a1911b03913062ee42c8a907a8cba145	30db42cfba7ba391bfd2fcca9819cb4a
19981205	0000004	001	000003	1234567890	D	00000150000	30db42cfba7ba391bfd2fcca9819cb4a	34e3b0fb4a9711905bbd7e29e2548d5d
19981203	0000004	001	000003	1234567890	D	00000150000	34e3b0fb4a9711905bbd7e29e2548d5d	cda59761e360a83eecfb7e7ab09c6289

Figura 16: **Exemplo de encadeamento usando o algoritmo de *secure hash* MD5**

5.4. Formas de Validação de Transações

Com o uso do encadeamento de transações é possível fazer verificações no estado da base de dados com relação a integridade das transações tanto no momento em que as transações estão sendo realizadas quanto em momentos posteriores. São possíveis quatro tipos básicos de validação:

- **Validação Horizontal Imediata;**
- **Validação Vertical Imediata;**

- **Validação Horizontal Posterior;**
- **Validação Vertical Posterior.**

Veremos a seguir como podem ser implementadas cada uma delas.

- **Validação Horizontal Imediata**

  A validação horizontal imediata, executada no momento em que a transação está sendo efetuada, garante a integridade da transação após o seu tráfego na rede. É baseada na aplicação da função de *secure hashing* adotada para fazer a seguinte comparação:

  Hash Informado = Hashing(Dados da Transação + Hash Anterior Informado)

- **Validação Vertical Imediata**

  Esta validação, também realizada no momento em que a transação está sendo efetuada, permite que seja verificado se o contexto transacional entre o servidor e o cliente está correto. Ela baseia na inclusão pelo cliente, em cada nova transação, do valor *hash* da transação anterior. O servidor realiza uma comparação direta entre o valor informado pelo cliente e o valor armazenado nas suas bases para detectar se o contexto entre os dois está inconsistente através da seguinte comparação:

  Hash Anterior Armazenado = Hash Anterior Informado

Para maior eficácia, as duas formas de validação imediata podem ser realizadas juntamente com os mecanismos de integridade referencial realizados pelo gerenciador do banco de dados, através do uso conjunto de *triggers* e *stored procedures*. A Figura 17 exemplifica a definição de uma tabela usando o SGBD Oracle baseada no uso de *triggers* de bancos de dados e de uma *stored procedure* que calcula a função *de secure hashing*. Usaremos o mesmo caso de uma instituição financeira e de seus clientes, discutido anteriormente.

O *trigger* contido na Figura 17 é sempre ativado no momento da inclusão de registros novos na tabela **transacoes**, imediatamente antes do SGBD efetivar a operação, não importando de que forma a operação de inclusão foi solicitada. Para realizar as validações imediatas vertical e horizontal serão executadas as *stored procedures* contidas no *trigger* **ObtemHashAnterior**, para localizar o *hash* atual da última transação da conta corrente em pauta e **CalculaHashAtual**, que simplesmente implementa o cálculo do algoritmo de *secure hash* para o texto passado como parâmetro. Para as operações de alteração e exclusão, deveriam ser definidos *triggers* similares, com pequenas alterações. No caso da operação de alteração, o *hash* anterior a ser considerado não seria, necessariamente, o último do

contexto mas, sim, o imediatamente anterior à transação sendo alterada. Além disso, deveria ser verificado se o *hash* atual calculado continuava igual ao *hash* anterior da transação imediatamente posterior à transação sendo alterada, de modo a não quebrar o encadeamento. Com relação à operação de exclusão, para que o encadeamento não fosse quebrado o texto do *trigger* deveria cuidar para que apenas a última transação do contexto pudesse ser excluída. Em aplicações onde fosse necessário a realização de alterações ou exclusões de transações que não a última, o conceito de estorno deveria ser usado, ou seja, uma nova transação seria incluída para excluir ou alterar dados de uma anterior. Além de não quebrar o encadeamento esta estratégia permite que seja mantido um histórico real das operações realizadas.

DEFINIÇÃO DA TABELA

```
CREATE TABLE    transacoes (
data            DATE PRIMARY KEY,
numero          NUMBER( 7, 0 ) PRIMARY KEY,
historico       NUMBER( 3, 0 ) NOT NULL
                CONSTRAINT historicos_fkey
                REFERENCES historicos,
documento       NUMBER( 6, 0 ) NOT NULL,
contacorrente   NUMBER( 10, 0 ) NOT NULL
                CONSTRAINT contas_fkey
                REFERENCES contas,
tipo            VARCHAR2( 1 ) CONSTRAINT
                tipo_check CHECK (tipo IN ('C', 'D')),
valor           NUMBER( 11, 2 ) NOT NULL,
hashanterior    VARCHAR2( 32 ) NOT NULL,
hashatual       VARCHAR2( 32 ) NOT NULL );
```

DEFINIÇÃO DO TRIGGER DE INCLUSÃO DE REGISTROS

```
CREATE OR REPLACE TRIGGER validacao_imediata
BEFORE INSERT ON transacoes
FOR EACH ROW
DECLARE
   v_hashanterior     VARCHAR2( 32 );
   v_ hashatual       VARCHAR2( 32 );
BEGIN
   v_hashanterior := ObtemHashAnterior( :new.contacorrente );
   IF :new.hashanterior <> v_hashanterior THEN
      raise_application_error( -20201,
                        'Erro na validacao vertical imediata' );
   END IF;
   v_hashatual := CalculaHashAtual(
     :new.data || :new.numero || :new.historico || :new.documento
    || :new.contacorrente || :new.tipo || :new.valor
    || v_hashanterior );
   IF :new.hashatual <> v_hashatual THEN
      raise_application_error( -20202,
                        'Erro na validacao horizontal imediata' );
   END IF;
END;
```

Figura 17: **Exemplo de utilização de *triggers* de banco de dados em Oracle**

- **Validação Horizontal Posterior**

  Ao mantermos o valor de *hash* da transação no banco de dados, juntamente com as demais informações, temos a possibilidade de realizar, em qualquer momento, uma revalidação da integridade de todas as transações. Isto possibilita a detecção de alterações posteriores, de origem interna ou externa, nos dados das transações. Um simples processo *batch*, utilizando a mesma função de *secure hashing* referenciada no *trigger* do banco de dados, pode ser aplicado para esta verificação em todas ou em parte das transações.

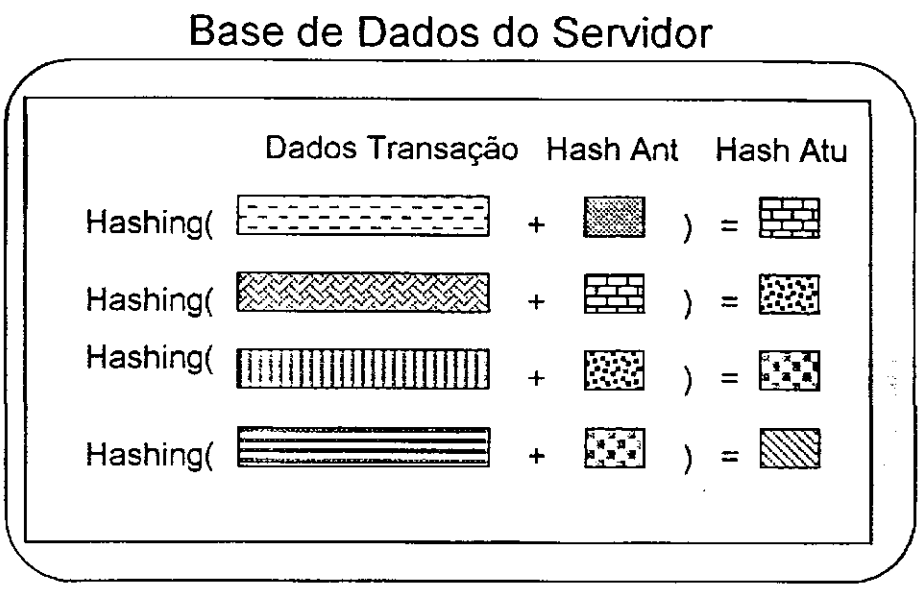

Figura 18: **Processo de Validação Horizontal Posterior**

- **Validação Vertical Posterior**

A validação vertical posterior, da mesma forma que a validação vertical imediata, permite que o contexto transacional entre o servidor e o cliente seja verificado. Um processo *batch*, executado a qualquer momento pelo servidor, compara para cada transação realizada por um determinado cliente, o valor de *hash* de uma transação com o valor de *hash* anterior da transação imediatamente posterior. Se o encadeamento de transações for quebrado em qualquer momento, através da manipulação da base de dados diretamente, será detectado aqui.

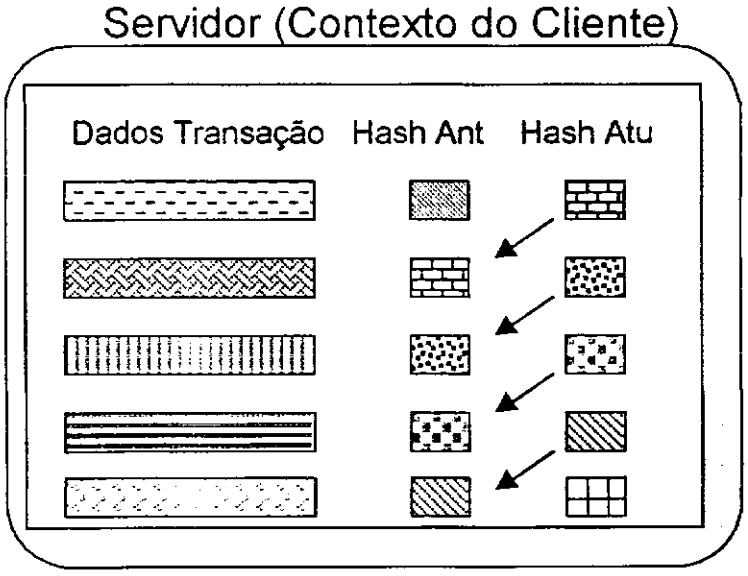

Figura 19: **Processo de Validação Vertical Posterior**

5.5. Detecção de Quebras de Encadeamento (Exceções)

Quando uma das formas de validação descobre que o encadeamento foi quebrado fica caracterizada uma situação de exceção. Dependendo do momento em que a quebra acontece, podemos determinar que ações devem ser tomadas:

- **Cliente Inválido:** Se a comparação do contexto não conferir no momento da validação horizontal imediata, trata-se de um cliente inválido ou de um erro de transmissão. Neste caso, simplesmente ignorar a transação pode ser suficiente.
- **Tentativa de Fraude:** Se o problema foi detectado no momento da validação vertical imediata, está caraterizada uma perda de sincronismo do contexto com o cliente. Embora seja possível simplesmente desprezar a transação, uma ação mais adequada seria bloquear a realização de transações com aquele cliente. Aqui temos uma situação mais séria que exige maiores cuidados, pois tanto pode ser um intruso tentando realizar uma fraude após ter descoberto as senhas de um determinado cliente, quanto um cliente legítimo realizando uma transação após um intruso ter sido bem sucedido e ter descoberto até mesmo o seu contexto. Embora em menor escala, este tipo de quebra também pode ser causada pela ocorrência de algum erro durante a realização da última transação. Em todos os casos, um procedimento de re-sincronização, inclusive por outro meio que não digital, deve ser realizado para este cliente.
- **Fraude ou Falha Posterior:** Durante as validações horizontais e verticais realizadas posteriormente, é possível que seja detectada alguma inconsistência nas transações já realizadas. Três causas principais podem ser responsáveis por este tipo de quebra do encadeamento: falhas de *hardware* ou *software*, erros humanos ou fraudes diretas na base de dados, de origem interna ou não. Após a detecção, a ação de reconstrução vai depender de caso para caso. Além da solução imediata, a causa deve ser identificada para que sejam tomadas medidas preventivas contra novas ocorrências e, caso desejado, medidas punitivas também (investigação, inquérito etc).

5.6. Encadeamento com *Hash* Derivado

Outra forma de se realizar a troca de mensagens entre os módulos Cliente e o Servidor sem que o valor de *hash* calculado pelo cliente trafegue pela rede é através da adoção de uma técnica conhecida como prova de conhecimento zero (*zero-knowledge proof*) [WAY97]. Com esta técnica, ao invés do cliente enviar o valor de *hash* real que autentica o encadeamento de transações trocadas com o servidor, ele calcula e envia um outro valor derivado do *hash* original, mas que não permita a sua dedução caso a mensagem seja interceptada por algum

intruso. Com este valor derivado do *hash* original, o módulo Servidor repete os passos realizados pelo Cliente e verifica a autenticidade da transação. Para calcular o valor derivado do *hash* de contexto atual, pode ser aplicada novamente a mesma função de *secure hashing* já utilizada para cálculo do valor de *hash* atual do contexto:

Hash Atual = Hashing(Dados da Transação + Hash Anterior)

Hash Derivado = Hashing(Hash Atual)

Como o valor a ser enviado na transação passa a ser o valor derivado e não o original, a característica de não inversibilidade da função de *secure hashing* garante que mesmo que o valor derivado seja capturado ele não poderá ser usado pelo intruso para se infiltrar no contexto entre o servidor e o cliente. Para melhorar o entendimento, vamos visualizar este procedimento alternativo aplicado aos passos do encadeamento de transações apresentados anteriormente:

- **1º Momento: Cliente monta a Transação Contextualizada com o Hash Derivado**

  Nesta variação, ao compor uma nova transação, o módulo cliente da aplicação deve aplicar **duas** vezes a função de *secure hashing* adotada sobre os dados específicos da transação e também sobre o valor de *hash* da última transação realizada. Obtém-se assim o valor de *hash* derivado para a nova transação que deve ser enviado juntamente com os dados da transação para o módulo servidor obedecendo aos critérios de segurança (rede e interna) implementados. Observe que nesta variação o valor do *hash* anterior não é enviado com a transação. Ele passa a ser um elemento implícito, que da mesma forma que o *hash* atual, nunca trafega na rede. Isto também é possível sem o uso do *hash* derivado. Em qualquer dos casos, são necessários, obviamente, pequenos ajustes também nos processos *batch* associados com as validações posteriores (vertical e horizontal) para refletir as mudanças se o *hash* anterior for usado de forma implícita, sem ser armazenado juntamente com a transação atual.

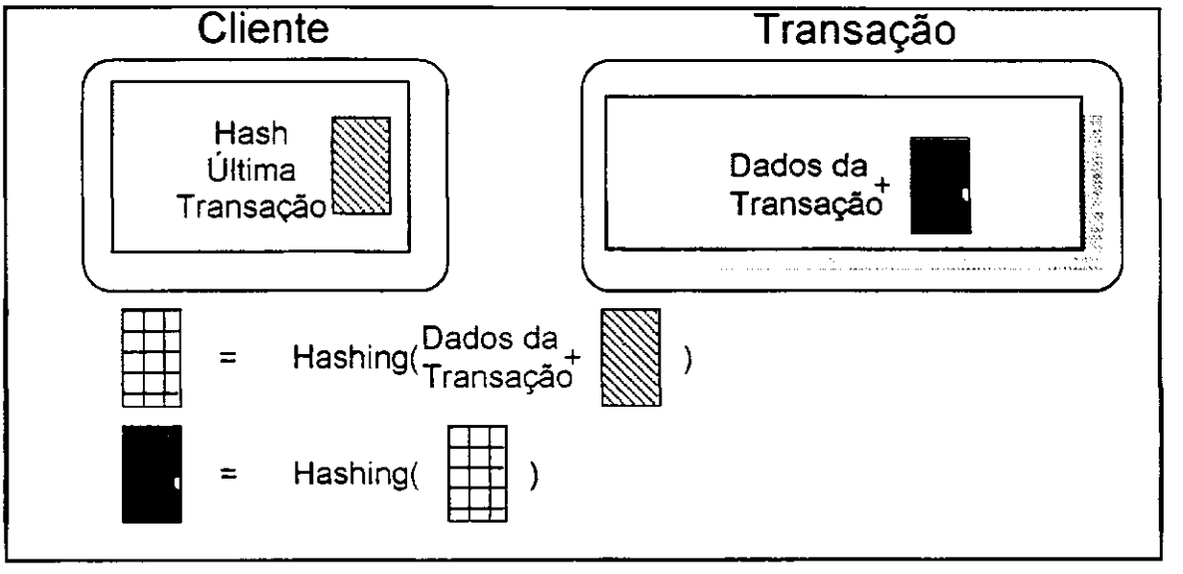

Figura 20: **Transação Contextualizada com Hash Derivado**

- **2º Momento: Servidor valida a Transação Contextualizada com Hash Derivado**

  Ao receber uma nova transação de um determinado Cliente, o Módulo Servidor deve, inicialmente, aplicar as técnicas de autenticação e validação previstas na segurança tradicional. Vencida esta etapa, o próximo passo é calcular o *hash* atual da transação usando os dados da transação e o *hash* da transação anterior armazenado. O servidor faz isto consultando a sua base de dados persistente e aplicando a função de *secure hashing* pré-acordada. Em seguida, o Servidor deve reaplicar a função de *secure hashing* sobre o valor de *hash* atual que calculou e verificar se é igual ao *hash* derivado incluído na transação.

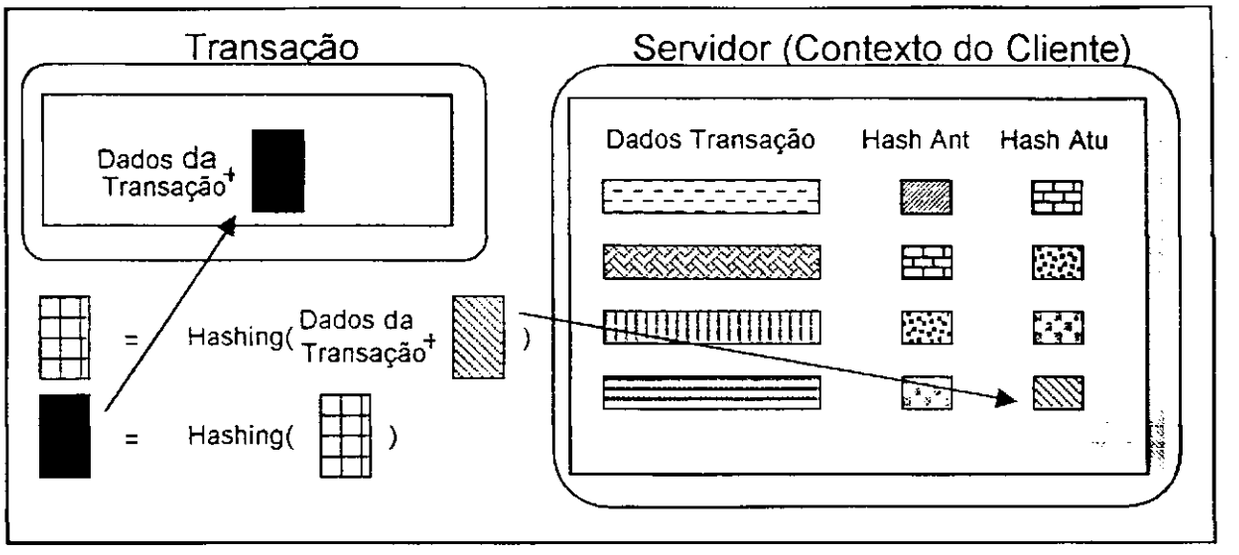

Figura 21: **Validação da Transação Contextualizada com Hash Derivado**

- **3° Momento: Servidor registra a Transação**

A princípio este passo se mantém inalterado, embora uma modificação seja possível: pode ser suprimido o armazenamento do *hash* anterior da última transação (vide destaque na Figura 22) e, ao invés de uma referência explícita, pode ser utilizada uma referência implícita, onde mesmo ainda sendo utilizado para o cálculo do *hash* atual da transação, o *hash* anterior seria obtido a partir do *hash* atual da última (imediatamente anterior) transação realizada. Observe que para o registro na base de dados permanente foi usado o *hash* original e não o *hash* derivado.

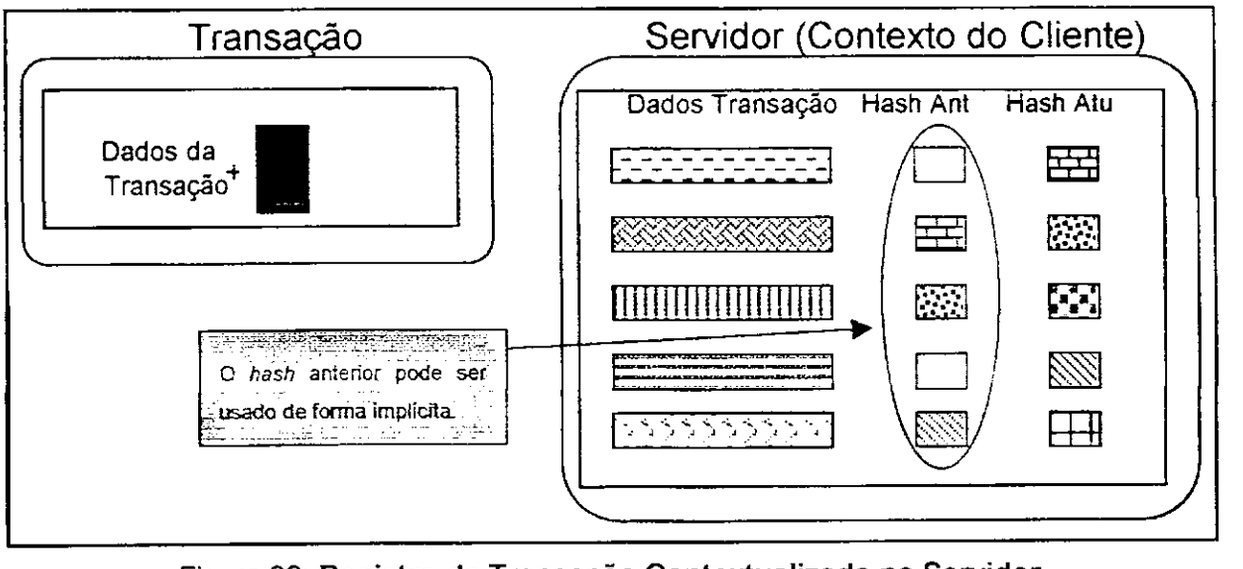

Figura 22: **Registro da Transação Contextualizada no Servidor**

- **4° Momento: O Cliente atualiza dados sobre o Contexto**

  Também não há variações neste passo a ser realizado pelo Cliente. Para ser armazenado na camada persistente, usa-se o novo valor de *hash* original obtido com a aplicação da função de *secure hashing* uma única vez.

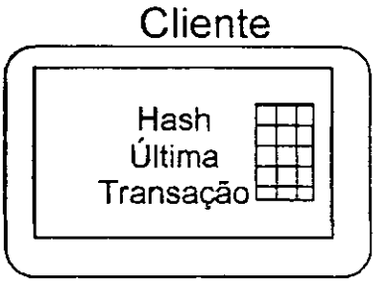

Figura 23: **Registro do Contexto pelo Cliente**

A Figura 24 ilustra o mesmo exemplo de encadeamento de transações para o caso da aplicação bancária, também usando o algoritmo MD5 para o cálculo do *secure hash* só que agora usando a técnica de *hash* derivado. Observe que os valores de *hash* atual e *hash* anterior, embora ainda necessários e aplicados para o cálculo do *hash* derivado não fazem mais parte da transação enviada pela aplicação cliente através da rede.

DATA	NÚMERO	HIST.	DOCTO	CONTA	TIPO	VALOR	HASH DERIVADO
19981201	0000001	001	000001	1234567890	D	00000100000	b9213a3054ac499f36a7874beae13acf
19981203	0000023	002	000002	1234567890	C	00000300000	7bbaa35a741732f617e09d29a3bed5ee
19981205	0000004	001	000003	1234567890	D	00000150000	8d8c2138992716e15ef9d78a764b4a44
19981203	0000004	001	000003	1234567890	D	00000150000	7f11baf5373a5dc43b248ffee3cbaca9

Figura 24: **Exemplo de encadeamento com *hash* derivado usando MD5**

Ainda baseado no conceito de *hash* derivado, é possível que a aplicação cliente também autentique a aplicação servidor. Basta incluir, no protocolo de mensagens trocadas na realização de uma transação, uma mensagem-desafio, que pode ser um número randômico, proposto pelo cliente para o servidor, e que deve ter como resposta um valor de *hash* obtido com a aplicação da função de *secure hashing* sobre o texto do desafio concatenado com o valor de *hash* anterior do cliente, que o servidor resgataria da sua base de dados

permanente. Este desafio inibe ataques baseados em reemissão (*replay*) de mensagens capturadas por servidores falsos.

Do Cliente para Servidor:	**Desafio = Random()**
Do Servidor para o Cliente:	**Resposta = Hashing(Hash Anterior + Desafio)**

5.7. Vantagens da Contextualização entre Aplicações

A criação deste tipo de contexto proporcionado pelo encadeamento de transações tem duas vantagens principais, relacionadas com a segurança da aplicação:

- **Detecção de Fraudes**: Seria necessário a captura das senhas e das informações do contexto e também do conhecimento do protocolo praticado pela aplicação, para que um intruso pudesse se inserir em um determinado contexto se fazendo passar integralmente pelo cliente associado. Vamos supor, entretanto, que um intruso tenha a sorte ou a habilidade necessárias e consiga burlar tanto o sistema de segurança tradicional como também o controle de encadeamento, produzindo uma ou mais transações ilegítimas mas corretas no momento certo. Para a fraude ser perfeita, o fraudador teria ainda que substituir o contexto do cliente pelo novo contexto, caso contrário, na próxima vez em que o cliente legítimo fosse fazer uma transação encontraria um contexto diferente, gerando assim uma situação de exceção (Figura 25). Isto é muito mais difícil, pois a quebra de informações sigilosas está mais ligada a leitura ou captura ilegítima de dados do que a sua gravação ou substituição.

  A análise da exceção, baseada na reclamação do cliente legítimo, permitiria não somente detectar e interromper a fraude, mas também identificar que transações foram feitas pelo intruso. Para isto, bastaria apenas levantar qual o último *hash* armazenado pelo cliente legítimo, todas as transações realizadas daquele ponto em diante seriam falsas.

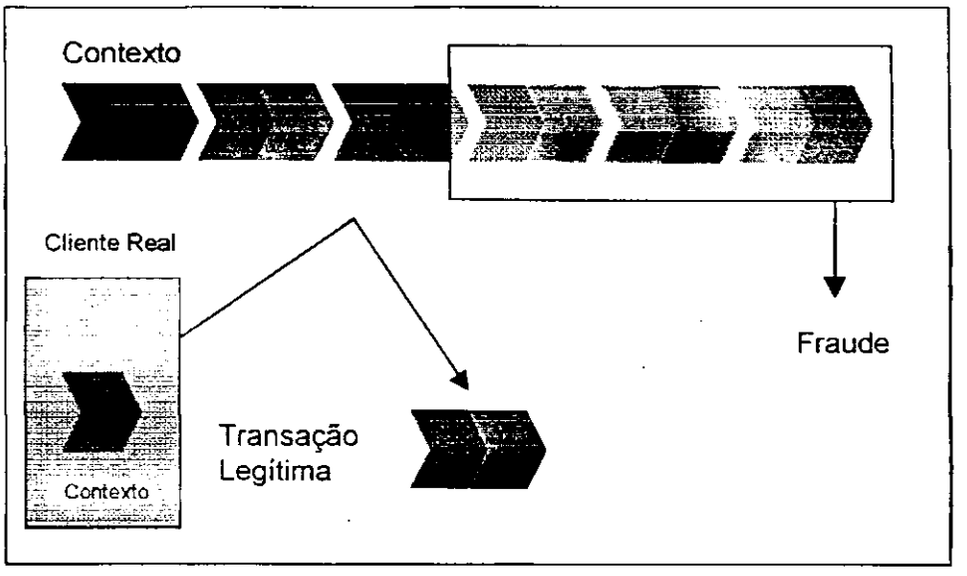

Figura 25: **Detectando fraudes com o encadeamento de transações**

- **Prevenção de Fraudes:** Como existe um momento único, na seqüência de transações trocadas entre as aplicações cliente e servidor, em que a validação de uma transação montada pelo cliente é possível, um mecanismo de segurança adicional é criado de forma natural. Desta forma, mesmo que as outras camadas de segurança falhem ou sejam burladas, permitindo, em um caso extremo, que um intruso falsifique corretamente uma transação com base na captura de informações sobre um determinado contexto, é possível que o *hash* esperado para a próxima transação já tenha mudado, pois ao contrário de uma senha ou chave privada, ele é dinâmico. Dependendo da freqüência com que as transações são realizadas entre o cliente e o servidor, em determinadas aplicações as informações sobre o contexto podem ficar obsoletas muito rapidamente, envelhecendo antes que possam ser usadas. O uso do encadeamento de transações diminui a criticidade da guarda de informações sigilosas como senhas e chaves privadas, pois elas sozinhas não permitem a autenticação necessária para a realização de transações, requerendo o uso complementar das informações associadas ao contexto que, por sua vez, mudam constantemente, numa freqüência proporcional à quantidade de transações realizadas. Esta característica do encadeamento de transações reduz o valor das senhas estáticas e, proporcionalmente, também diminui o esforço necessário para protegê-las, o que atende bem ao **Princípio da Proteção Adequada**. A ação a ser tomada na detecção de uma tentativa de fraude pode variar, desde simplesmente rejeitar a transação, apenas registrando a sua ocorrência, até mesmo o bloqueio do contexto até uma averiguação mais precisa do ocorrido. A Figura 26 ilustra como a existência do contexto permite recusar as tentativas de fraudes por

reconhecer as transações diferentes daquela esperada na ordem formada pelo contexto[23].

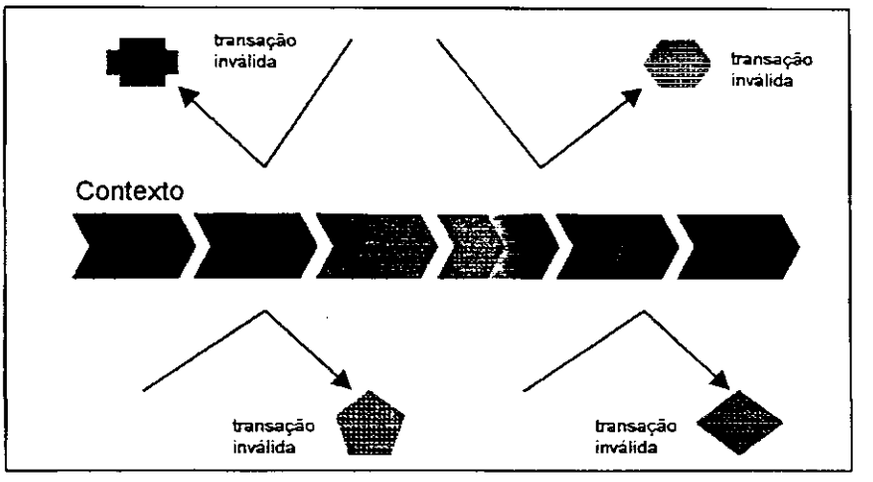

Figura 26: **Prevenindo fraudes com o encadeamento de transações**

Tais vantagens também são igualmente eficazes para a detecção e prevenção de erros humanos e falhas de *hardware* e *software*, e não apenas para o controle de fraudes.

5.8. Autenticando Usuários de Acesso Remoto: Um Exemplo Completo

A seguir, vamos construir uma aplicação distribuída exemplo voltada para a autenticação de conexões de acesso remoto a redes corporativas ou Internet, aplicável tanto para validar funcionários de empresas quanto usuários de provedores de acesso comerciais e usando a técnica de encadeamento de transações discutida anteriormente.

5.9. O Contexto: Acesso Remoto à Internet

A necessidade de acessar aplicações e bases de dados de um computador a partir de localidades remotas, tem sua origem na década de 60, quando os primeiros sistemas *time-sharing* e multi-usuários começaram a surgir. A evolução das tecnologias de comunicação de dados, computadores e o crescimento das redes de computadores, mudou a definição de acesso remoto [GAM86]. Neste novo contexto, acesso remoto tornou-se sinônimo de acesso aos recursos de uma rede através de um modem e de uma linha telefônica, discada ou não, por qualquer usuário, sistema ou servidor não diretamente conectado à rede. Nos dias de

[23] Esta possibilidade de detecção de tentativas de acessos ilegítimos pode ser associada com a proposta de sistemas sobreviventes [EFL97], principalmente nos requisitos de resistência e reconhecimento de ataques.

hoje, o acesso remoto a redes corporativas e Internet a partir de computadores pessoais (PCs) e computadores portáteis (*notebooks*) tornou-se bastante comum. Os tipos de acesso remoto mais utilizados são:

- **Nó remoto:** Um PC remoto ou estação de trabalho que disca e participa como um nó totalmente funcional da rede principal. Através do nó remoto de acesso, as aplicações residem no próprio PC remoto, com os recursos da rede disponíveis de forma transparente, parecendo serem locais. A comunicação na linha acontece somente quando é necessário transferir dados à localidade central. Um dos benefícios da utilização dos nós remotos é que os usuários não precisam ser treinados novamente, pois utilizam as mesmas interfaces disponíveis quando diretamente conectados à rede principal. Além disso, a natureza do acesso cliente-servidor dos nós de acesso, permite ao usuário remoto acessar sistemas de arquivos, correio eletrônico, impressoras ou qualquer outro recurso da rede. O nó remoto exige a utilização de *modems* de alta velocidade. Esta forma de acesso é a mais utilizada e deve estabelecer-se como de uso padrão, devido a alta capacidade de processamento dos PCs atuais e da evolução tecnológica dos modems e estruturas de comunicação em geral.

- **Controle Remoto:** Um PC remoto que disca e toma o controle de um PC da rede principal. Quando se utiliza a modalidade de controle remoto de acesso, o usuário remoto não executa as aplicações localmente em seu PC. As aplicações são executadas em um servidor ligado na LAN principal. Apenas atualizações de tela e teclado são passados através da linha. A vantagem do controle remoto é que este minimiza a quantidade de dados trafegados através da linha, sendo então adequado com o uso de *modems* de baixa velocidade e PCs de pequena capacidade de processamento. A desvantagem é que como o PC remoto controla as ações do servidor destino através de um *software* próprio, o usuário do PC deverá ser treinado numa nova forma de acessar os recursos da rede. Esta modalidade de acesso remoto também é muito utilizada para suporte a distância, pois permite uma visualização exata por parte do técnico das circunstâncias que possam estar causando um mal funcionamento no computador do usuário.

Para atender tal demanda de acesso remoto, surgiram tecnologias voltadas para **Servidores de Acesso Remoto** (*Remote Access Server - RAS*). Dois "formatos" de montagem de RAS passaram a ser utilizados e ainda são válidos hoje em dia:

- **Servidores de Comunicação**: servidores, normalmente com arquitetura PC, equipados com placas multi-seriais[24] inteligentes com múltiplas portas de comunicação, executando sistemas operacionais como UNIX ou Windows NT e que se ligam a um barramento Ethernet TCP/IP através de uma placa de rede.

- **RAS externo independente** ou ***terminal server*** : equipamento específico para acesso remoto, que também se liga a um barramento Ethernet TCP/IP e possui um certo número de portas de comunicação, normalmente de 8 a 32 portas. Cada porta serial RS-232 do RAS deve ser conectada a uma linha telefônica associada e também a um modem[25].

Nos dois casos, são necessários serviços de **autenticação** e, eventualmente, mecanismos de **bilhetagem** de usuários. Por **autenticação**, entende-se a necessidade de permitir ou não o acesso de um usuário remoto aos recursos de uma rede. A autenticação requer a existência de uma base de dados onde estão cadastrados os usuários, suas senhas e os recursos que estão disponíveis para os mesmos. Esta base de dados pode ser local ao Servidor de Acesso Remoto ou situado remotamente, em um servidor da rede (denominado Servidor de Autentîcação). Já a **bilhetagem** é necessária para contabilização de uso dos recursos por parte dos usuários, seja para cobrança, seja para fins estatísticos. A informação mais comumente utilizada é o tempo que o usuário ficou conectado. Porém, outras informações podem ser utilizadas: número de acessos, total de pacotes recebidos ou transmitidos, total de bytes recebidos ou transmitidos. A comunicação entre o Servidor de Acesso Remoto e o Servidor de Autenticação se dá através de protocolos definidos especificamente para essa função. Os protocolos de autenticação e bilhetagem mais comuns são o **RADIUS** [LUC97] e **TACACS** [CIS95].

A grande penetração da Internet nos mais variados setores da sociedade foi um dos principais motivos para a explosão da demanda por acesso remoto, tanto com relação a serviços quanto a produtos associados. Com os PCs cada vez mais rápidos e capazes de processar imagens e gráficos, a necessidade de velocidades maiores nos *modems* criou

---

[24] Embora também seja possível usar as portas seriais normais do computador ou *placas fax-modems*, as placas multi-seriais permitem estender a capacidade de conexão para quantidades similares às oferecidas por equipamentos RAS específicos.

[25] A dificuldade de se montar e manter uma estrutura com muitas linhas telefônicas e modems levou a proposta de um novo formato de RAS: "Channelized T1/E1" RAS, ou Servidores de Acesso Remoto T1/E1 Fracionados, baseados na tecnologia de ISDN.

uma competição maior entre os fabricantes e os padrões V34 (28.800 bps), V34+ (33.600 bps) e V90[26] (56.600 Kbps) apareceram. A partir de 1995 observou-se uma corrida em quase todos os países do mundo para se oferecer acesso discado para a Internet. Um PC, um *modem* e uma linha telefônica era tudo que o usuário precisava para acessar a Internet.

Surgiu então um mercado novo, repleto de provedores comerciais de serviços para a Internet (*Internet Service Providers - ISPs*). Os ISPs[27], atuando como RAS através da disponibilização de uma estrutura de comunicação, permitem que os seus assinantes se conectem à Internet, pagando uma determinada tarifa por isso.

5.10. O Problema: Inibir o Acesso Indevido

Os *hackers* e intrusos do gênero são usuários compulsivos de computadores e costumam ficar conectados por muito tempo e com muita freqüência para realizarem seus ataques. Uma das providências iniciais tomadas por eles é descobrir maneiras de não precisar pagar por isto, burlando desde os provedores de acesso até as companhias telefônicas. Como os provedores de acesso baseiam a autenticação de seus usuários, quase que na sua totalidade, no modelo usuário/senha, é muito fácil para os *hackers* obterem estas senhas, seja com ferramentas de quebra por força bruta, seja através de engenharia social, obtendo-as de forma ardilosa dos próprios usuários. Uma vez que uma senha de usuário é descoberta, ela é rapidamente divulgada na própria Internet entre os *hackers*, numa cadeia de favorecimentos mútuos, que inclui também o compartilhamento de outros segredos e técnicas de invasão.

Por isto, vários usuários de provedores são surpreendidos com cobranças de valores extremamente altos de faturas com os provedores, ocasionados pelo uso indevido de suas contas por *hackers*. Este tipo de situação causa transtorno e aborrecimentos tanto para o usuário, que nem sempre entende bem o problema e culpa o provedor de acesso, quanto para o provedor, que termina arcando com o prejuízo para não abalar o seu relacionamento com o cliente. Outro problema para os provedores é que nem sempre uma reclamação de um usuário é verídica, às vezes, tratam-se de usuários agindo de má fé e usando este tipo de ocorrência para tentar reduzir a sua fatura mensal. Também prejudicadas pelo uso

---

[26] Para esta velocidade, diversas propostas ainda tentam se estabelecer como padrão sendo a norma V90 uma das mais difundidas.

[27] Além do RAS, os ISPs oferecem diversos outros serviços como confecção e hospedagem de *homepages*, conexão dedicada à Internet para empresas, domínios virtuais etc.

indevido de senhas são as empresas, que usam algumas contas básicas nos provedores e que são compartilhadas por vários funcionários. Neste casos, com freqüência, há vazamento para familiares e amigos, que passam a usar estas contas para uso pessoal.

Um agravante recente, principalmente para os provedores, é o surgimento no mercado da modalidade de acesso sem limite de horas, o que facilita enormemente o uso de senhas roubadas. Nestes casos, não há mais nenhuma facilidade de detecção de fraudes pois, o usuário fica totalmente indiferente a qualquer quantidade de horas que sejam apuradas na sua conta, já que não há mais limite e ele vai pagar sempre o mesmo valor. O provedor, por sua vez, mesmo se deparando com quantidades absurdas de horas, não pode cobrar nenhum valor adicional do usuário, cabendo apenas tentar detectar e inibir o uso simultâneo, ou seja duas conexões do mesmo usuário ao mesmo tempo, ou controlar a origem da chamada, o que não é tão simples.

Uma das formas de abordar este problema é através da adoção de mecanismos *de one-time passwords*, baseados em tecnologias especiais de *hardware* como o *SecureId Card* [GS97]. Com tal equipamento, que produz um valor especial cada vez que é ativado e que deve ser usado como senha pelo usuário, o provedor de acesso pode eliminar quase que totalmente as fraudes. O principal problema desta solução é o seu alto preço, o que limita o seu uso em larga escala. Existe, realmente, a necessidade de uma alternativa de baixo custo e que possa ser facilmente implantada, não importando a tecnologia dos servidores de acesso remoto ou o protocolo usado para autenticação e bilhetagem.

O problema de autenticação de usuários de provedores de acesso é perfeito para exemplificar o uso da técnica do encadeamento de transações, pois envolve um ambiente distribuído, é associado com a Internet e a aplicação resultante é simples, voltada quase que exclusivamente para a autenticação de usuários. Vamos ver como poderíamos construir uma solução para este problema a seguir.

5.11. Uma Solução com Encadeamento de Transações

O uso do encadeamento de transações para a autenticação de usuários de acesso remoto pode minimizar o problema de fraudes no uso indevido de contas, pois o candidato à nova conexão precisaria provar que realizou também a anterior e assim por diante, em uma validação adicional. O principal fator para o incremento da segurança é que ao invés de contarmos apenas com uma autenticação baseada em identificação/senha, passaremos a contar também com uma espécie de *one-time password* lógico, solicitado como desafio pelo provedor. Este valor, que é o *hash* do encadeamento das conexões anteriores, mudará constantemente e será válido apenas para uma determinada conexão. Após a sua

utilização, o valor representativo do contexto do usuário com o provedor perderá imediatamente a sua validade. Desta forma, se um *hacker* conseguisse obter a identificação e a senha de um usuário de um provedor, não conseguiria se conectar, a menos, que conseguisse obter também o valor atual do contexto[28]. Supondo que o *hacker* conseguisse também este valor e realizasse uma ou mais conexões se fazendo passar por um determinado usuário, ainda assim teríamos vantagens por usar o encadeamento de transações. Na primeira vez que o usuário legítimo tentasse realizar uma conexão, a mesma seria rejeitada, pois o contexto armazenado pelo usuário legítimo já estaria desatualizado[29]. Esta exceção poderia desencadear diversas ações como alertas para o suporte ou, até mesmo, o bloqueio da conta do usuário para averiguação. Também seria possível identificar com precisão que conexões foram realizadas por um *hacker*, bastando apenas identificar o valor de contexto mantido pelo usuário legítimo, que indicaria a última conexão válida. Todas as posteriores a esta teriam sido feitas pelo *hacker*. Teríamos então proteção para o provedor e também para os usuários legítimos, com a minimização das perdas até mesmo em caso extremos de vazamento tanto das senhas quanto das informações sobre os contextos de conexão de cada usuário. É importante frisar que o uso do encadeamento de transações não substitui a autenticação já utilizada pelo provedor, mas soma-se a ela, complementado-a em busca de maior segurança.

A aplicação exemplo do uso do encadeamento de transações, que é relativamente fácil de ser implementada, é composta de dois módulos: o módulo PROVEDOR e o módulo USUÁRIO. A comunicação entre eles é baseada no modelo cliente/servidor e construída com *sockets* [CS91] do TCP/IP [COM91], onde o módulo USUÁRIO representa o processo servidor e o módulo PROVEDOR representa o processo cliente. Esta aparente "inversão" do modelo se deve ao fato de que o módulo PROVEDOR sabe, com exatidão, quando o canal de comunicação está estabelecido, sendo possível realizar a troca de mensagens de imediato sem haver a necessidade de temporizadores para isto. Do ponto de vista da segurança, eles realizam uma comunicação ponto-a-ponto, sem intermediários. Apesar disso, foi também adotado um algoritmo de criptografia simétrico (Figura 27) que, embora pouco seguro, é rápido, fácil de implementar e atende bem às necessidades da aplicação, sendo sempre possível usar a técnica do *hash* derivado em ambientes mais críticos. Para o

[28] Note que para obter isto o intruso precisa invadir dois sistemas, o servidor do provedor e o computador do usuário.

[29] A menos que o intruso também altere a informação de contexto do usuário, o que nesta aplicação é bastante improvável.

armazenamento do contexto do clientes foi usado o próprio registro do Windows e como função de *secure hashing*, foi adotado um algoritmo de CRC polinomial.

```
function Criptografa(pTransacao, pChave: String; pCrypto:
pchar): integer;
var i,x,y: shortint;
begin

 x := 1; y := +1;
 for i:=1 to Length(pTransacao) do
  begin
   pCrypto[i] := Char(Byte(pTransacao[i]) xor
Byte(pChave[x]));
   pCrypto[i] := Char(Byte(pCrypto[i]) shl 4 + Byte(pCrypto[i])
shr 4);
   if x = 1 then
    y := +1
   else if x = Length( pChave ) then
    y := -1;
   x := x + y;
  end;
 Criptografa := Length( pTransacao );

end;
```

```
function Decriptografa(pTransacao, pChave: String): String;
var i,x,y: shortint;
begin

 x := 1; y := +1;
 for i:=1 to Length(pTransacao) do
  begin
   pTransacao[i] := Char(Byte(pTransacao[i]) shl 4 +
Byte(pTransacao[i]) shr 4);
   pTransacao[i] := Char(Byte(pTransacao[i]) xor
Byte(pChave[x]));
   if x = 1 then
    y := +1
   else if x = Length(pChave) then
    y := -1;
   x := x + y;
  end;
 Decriptografa := pTransacao;

end;
```

Figura 27: **Criptografia Simétrica (em Delphi)**

A troca de mensagens entre eles obedece a um protocolo simples em que o módulo USUÁRIO, ao ser ativado pelo usuário e após se estabelecer como servidor de um serviço de rede, solicita uma identificação para acessar a sua base local e recuperar o valor do contexto do usuário associado com a identificação fornecida e uma chave de acesso (PIN), usada para decriptografar o valor do contexto para a sua forma natural (Figura 28). Em seguida, dispara uma tentativa de conexão com o provedor de acesso, utilizando-se os recursos normais de Rede Dial-Up do WINDOWS para isto (Figura 29) e fica em um estado de espera, aguardando pelo desafio do módulo PROVEDOR.

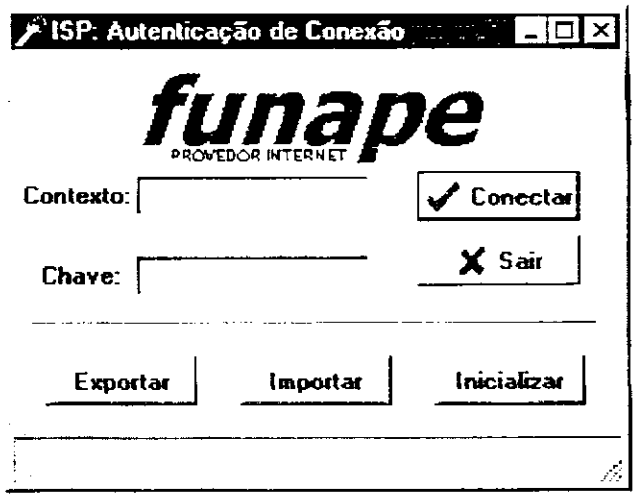

Figura 28: **Início do Processo de Autenticação**

O módulo PROVEDOR, por sua vez, só é ativado quando a comunicação PPP[30] é ativada pelo Servidor de Acesso Remoto (*RAS - Remote Access Server*). Neste momento, ele recebe algumas informações como o endereço IP do módulo USUÁRIO (a porta usada pelo serviço já é pré-estabelecida), o nome de *login* do usuário que identifica o contexto, o número de *tty* e a porta do servidor de comunicação utilizada etc. De posse de tais dados, o módulo PROVEDOR abre uma sessão TCP/IP com módulo USUÁRIO, monta e envia um desafio para ele. Fica, dentro de um determinado limite de tempo (*timeout*), aguardando a resposta do desafio. Se a resposta chegar a tempo, o módulo PROVEDOR recupera o *hash* anterior do contexto da sua base de dados persistente e calcula o *hash* atual, repetindo os mesmos passos que foram aplicados pelo módulo USUÁRIO. Caso o valor coincida, o módulo PROVEDOR devolve uma mensagem de aceitação para o módulo USUÁRIO, fecha a conexão TCP/IP e informa ao PPP que tudo está bem e que o canal de comunicação deve ser mantido. Em caso do *hash* enviado pelo módulo USUÁRIO não estar correto, ou de ter havido *timeout*, o módulo PROVEDOR devolve um valor de erro para o PPP, que derruba o canal de comunicação recém-estabelecido.

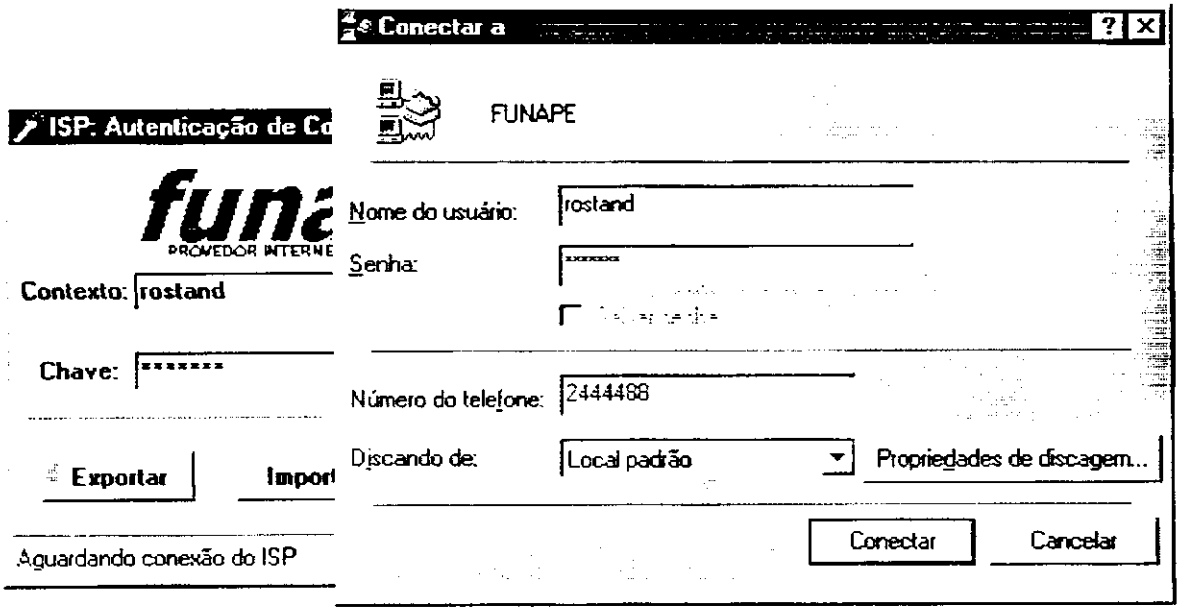

Figura 29: **Conexão ao Provedor de Acesso**

[30] *PPP – Point-to-Point Protocol* é um protocolo que permite a transmissão de datagramas sobre conexões seriais ponto-a-ponto [SIM92].

Discutiremos, a seguir, como são construídos os dois componentes distribuídos, no modelo cliente/servidor, necessários para implantarmos o encadeamento de transações de conexão em Servidores de Acesso Remoto – RAS:

- **Um Servidor de Validação de Contexto (Módulo USUÁRIO):** Este componente tem a função de permitir que usuários remotos forneçam a informação correta quando forem desafiados pelo módulo PROVEDOR, rodando no RAS, imediatamente após a autenticação convencional (*usename/password*) haver sido completada com sucesso e antes da liberação definitiva da conexão. A sua função principal é se estabelecer como um serviço de rede e aguardar um desafio, enviado pelo módulo PROVEDOR imediatamente após a comunicação no protocolo PPP ser estabelecida. De posse do teor do desafio, o módulo usuário recupera o valor do contexto anterior da sua base local (Figura 30), decriptografa-o usando a chave do contexto, concatena-o com o texto do desafio e aplica a função de *secure hash* (Figura 32). Em seguida, devolve o novo valor de *hash* calculado para o módulo PROVEDOR e aguarda a confirmação da conexão. No caso da conexão ser autenticada (Figura 31), o módulo USUÁRIO armazena o novo valor do *hash* na sua base local para ser usado na próxima conexão. Tanto pode ser usado a técnica de *hash* natural como a técnica do *hash* derivado. Para isto, basta apenas aplicar a função de CRC polinomial mais uma vez no *hash* calculado antes de enviá-lo para o módulo PROVEDOR.

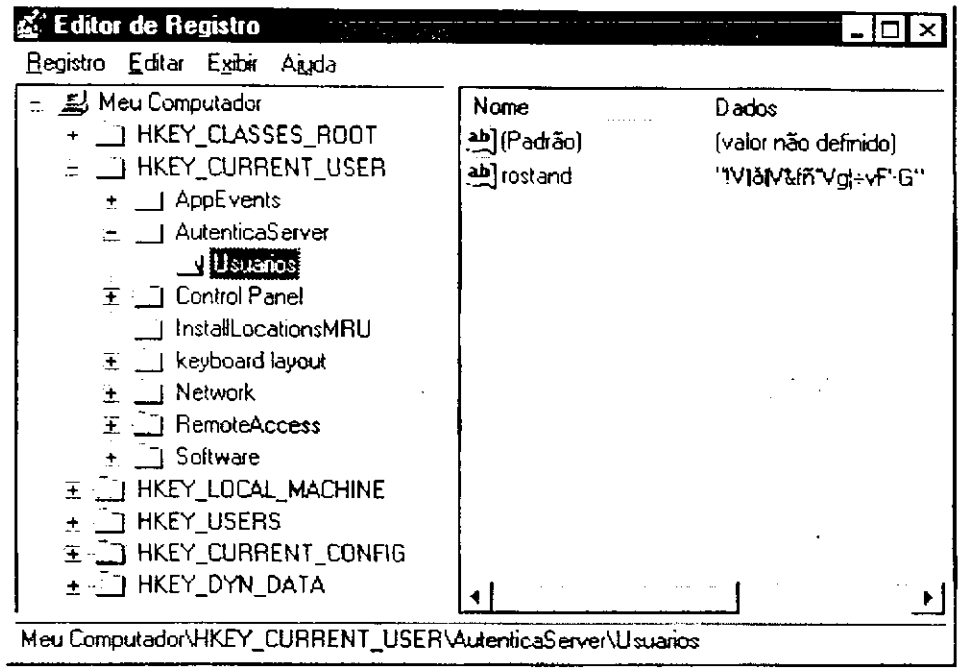

Figura 30: **Contexto armazenado no registro do Windows (criptografado)**

Com tal funcionalidade única e específica e sendo executado em ambiente Windows, o elemento servidor é um dos mais estáveis da solução, sendo praticamente independente do tipo de servidor de acesso remoto, forma de comunicação e protocolo de autenticação e bilhetagem utilizados pelo provedor de acesso. Tal caráter estanque permite que uma vez desenvolvido um protótipo de servidor para um determinado provedor ele possa ser facilmente reutilizado/adaptado para outros. O servidor aqui proposto é baseado na abordagem de *sockets* do TCP/IP, protocolo usado para se comunicar com o processo cliente desafiador.

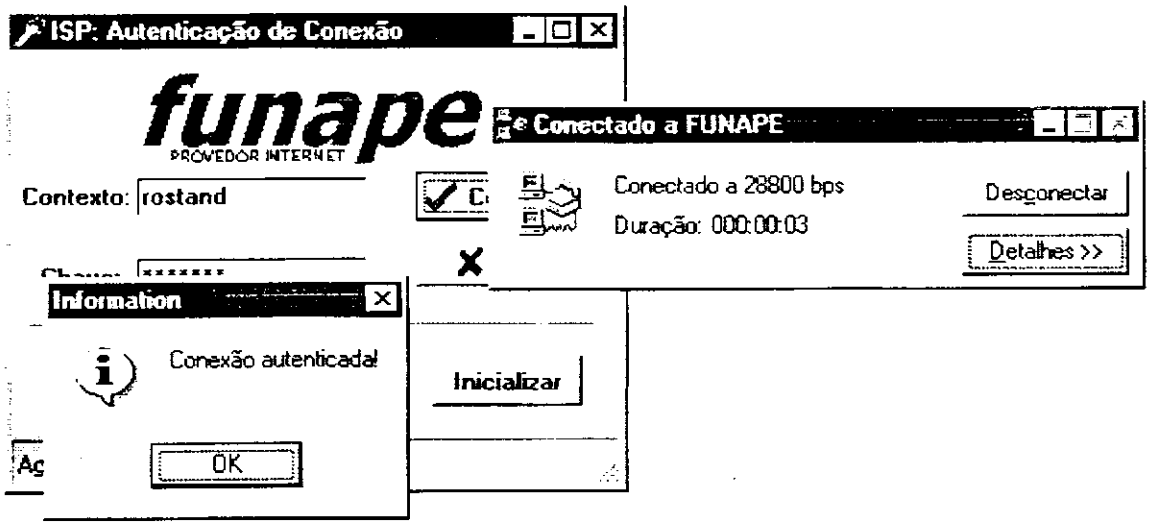

Figura 31: **Conexão Autenticada usando Encadeamento de Transações**

```
procedure Inicializa_Hash( pPoly: longint );
var i,j: shortint;
    soma: longint;
begin
    for i := 0 to 127 do
    begin
        soma := 0;
        for j := 7 - 1 downto 0 do
           if (i and (1 shl j)) > 0 then
                soma := soma xor (pPoly shr j);
        gTabelaCRC[ i ] := soma;
    end;
end;
```

```
function Calcula_Hash(pTransacao: pchar; pTam:
integer): String;
var  i: shortint;
     soma: longint;
     aux: string[ 11 ];
begin
    soma := 0;
    for i := 1 to pTam do
       soma    :=    (soma    shr    7)    xor
gTabelaCRC[((soma  xor  Byte(  pTransacao[i]))
and $7F) ];
    Str( soma, aux );
    Calcula_Hash := aux;
end;
```

Figura 32: **Função de CRC Polinomial (em Delphi)**

- **Um Cliente Desafiador de Contexto (Módulo PROVEDOR)**: Este componente tem a função de desafiar os usuários remotos para que forneçam uma determinada informação da forma correta imediatamente após a autenticação convencional (*usename/password*) haver sido completada com sucesso. Este componente é ativado automaticamente pelo *daemon* PPP [SIM92], logo após a conexão PPP estar ativada, recebendo algumas informações sobre a conexão, como endereços e portas utilizadas, bem como sobre o *login name* utilizado pelo usuário para se autenticar. A partir deste ponto, o módulo PROVEDOR tem como função principal se conectar com o serviço de rede estabelecido pelo módulo USUÁRIO, enviar-lhe um desafio e aguardar um determinado tempo por uma resposta válida. Caso o tempo limite de tempo transcorra ou a resposta recebida não seja válida, o módulo PROVEDOR determina que a conexão seja finalizada. Para determinar se o cliente respondeu corretamente ao desafio, o módulo PROVEDOR recupera o valor do contexto anterior da sua base permanente, concatena-o com o texto do desafio enviado e aplica a função de *secure hash* adotada (algoritmo de CRC polinomial). Em seguida, compara o novo valor de *hash* calculado com o valor devolvido pelo módulo USUÁRIO, que devem ser iguais. Neste caso, o módulo PROVEDOR deve armazenar o novo valor do *hash* na sua base permanente para ser usado na próxima conexão. Ocorrendo qualquer das hipóteses possíveis, a tentativa de conexão deve ser registrada em um arquivo de log, com todas as informações usadas para o desafio e também com a indicação se foi bem-sucedida ou não. Em casos de conexões não autenticadas (Figura 33), é neste momento que devem ser tomadas medidas de bloqueio da conta do usuário, se for esta a estratégia adotada.

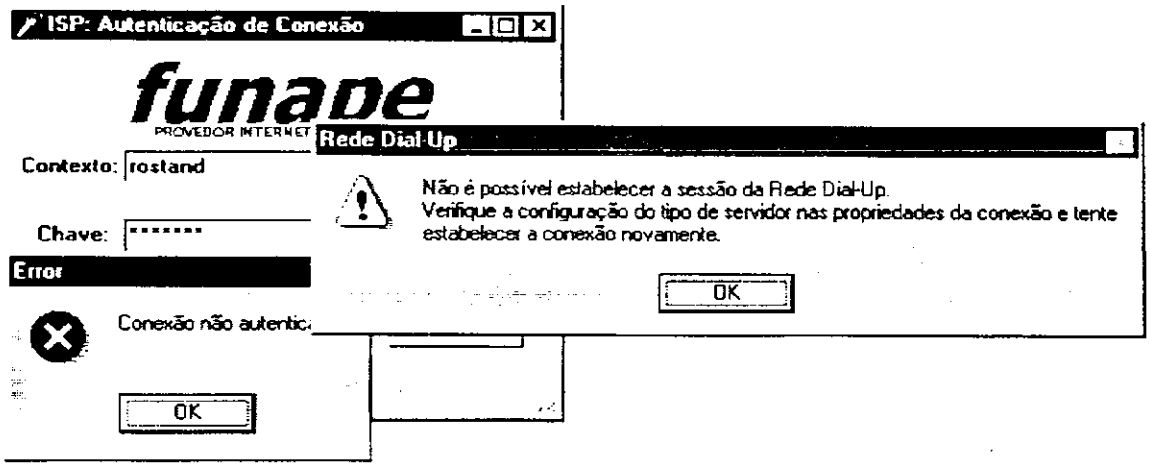

Figura 33: **Conexão não Autenticada usando Encadeamento de Transações**

Este componente, em si, também é relativamente estável e com possibilidade de reutilização em diversos ambientes. A principal adaptação necessária é identificar qual o ponto em que o protocolo de autenticação adotado pelo provedor (PAP/CHAP, RADIUS ou TACACS) finalizou o estabelecimento da conexão com o usuário remoto e, antes de liberá-la plenamente, ativar o módulo PROVEDOR com os parâmetros adequados para a validação final. A Figura 34 ilustra como foi feita esta adaptação para esta aplicação exemplo, onde o módulo PROVEDOR (**aut_client**) é ativado logo após o *daemon* PPP (**pppd**).

```
echo "Seu numero IP: $RMTIP"
echo "Meu numero IP: $MYIP"
echo "Nossa netmask: $NETMASK"
echo "Servidor de nomes (DNS): 150.165.249.1"
echo "Seu host name: lua"$PORTA
echo "Meu host name: terra"
echo "Nosso domain name: funape.ufpb.br"
echo "PPP ativado..."
rm -f /tmp/$RMTIP
echo $LOGNAME > /tmp/$RMTIP
chmod a+r /tmp/$RMTIP
/usr/sbin/pppd crtscts modem silent mtu 296 \
$MYIP:$RMTIP netmask $NETMASK disconnect '+++ATH0' \
lcp-echo-failure 1 lcp-echo-interval 600 &

desafio="`date` $LOGNAME $RMTIP $MYIP $PORTA"
retorno="CONEXAO AUTENTICADA"
/home/admin/rostand/tese/aut_client "$RMTIP" 6003 "$desafio" `cat
/home/admin/rostand/tese/$LOGNAME.aut` > /home/rostand/tese/$LOGNAME.aut
[ "$?" -eq "0" ] || {
retorno="ERRO DE AUTENTICACAO"
echo "$desafio $retorno" >> /home/admin/rostand/tese/$LOGNAME.log
logout
}
echo "$desafio $retorno" >> /home/admin/rostand/tese/$LOGNAME.log
```

Figura 34: **Trecho do *script* ppp_up que é ativado pelo PPP daemon**

A seguir, na Figura 35, temos uma visualização gráfica do fluxo de conexão e do papel de cada componente na autenticação de acesso remoto usando encadeamento de conexões. Logo após, na Figura 36, podemos observar um trecho de um *log* real de conexões

encadeadas, produzido pela aplicação exemplo. No log exemplificado, o módulo PROVEDOR recusou a última tentativa de conexão, numa simulação de uma situação de exceção, que tanto pode significar um intruso ou um usuário legítimo sendo rejeitado, ou seja, prevenção ou detecção de fraudes, respectivamente. Para melhorar a visualização e o entendimento, foi usado de forma explícita tanto o *hash* anterior quanto o *hash* atual. Duas variações, entretanto, são possíveis. A primeira é através do uso de *hash* anterior implícito, não fazendo parte da transação em si, mas ainda sendo utilizado no cálculo do *hash* atual. Outra variação, baseada no uso do *hash* anterior implícito e aplicável apenas na troca de mensagens entre o módulo USUÁRIO e o módulo PROVEDOR, é usar o *hash* derivado, o que aumenta a segurança da comunicação já que, em nenhum momento, o *hash* atual trafega pela rede.

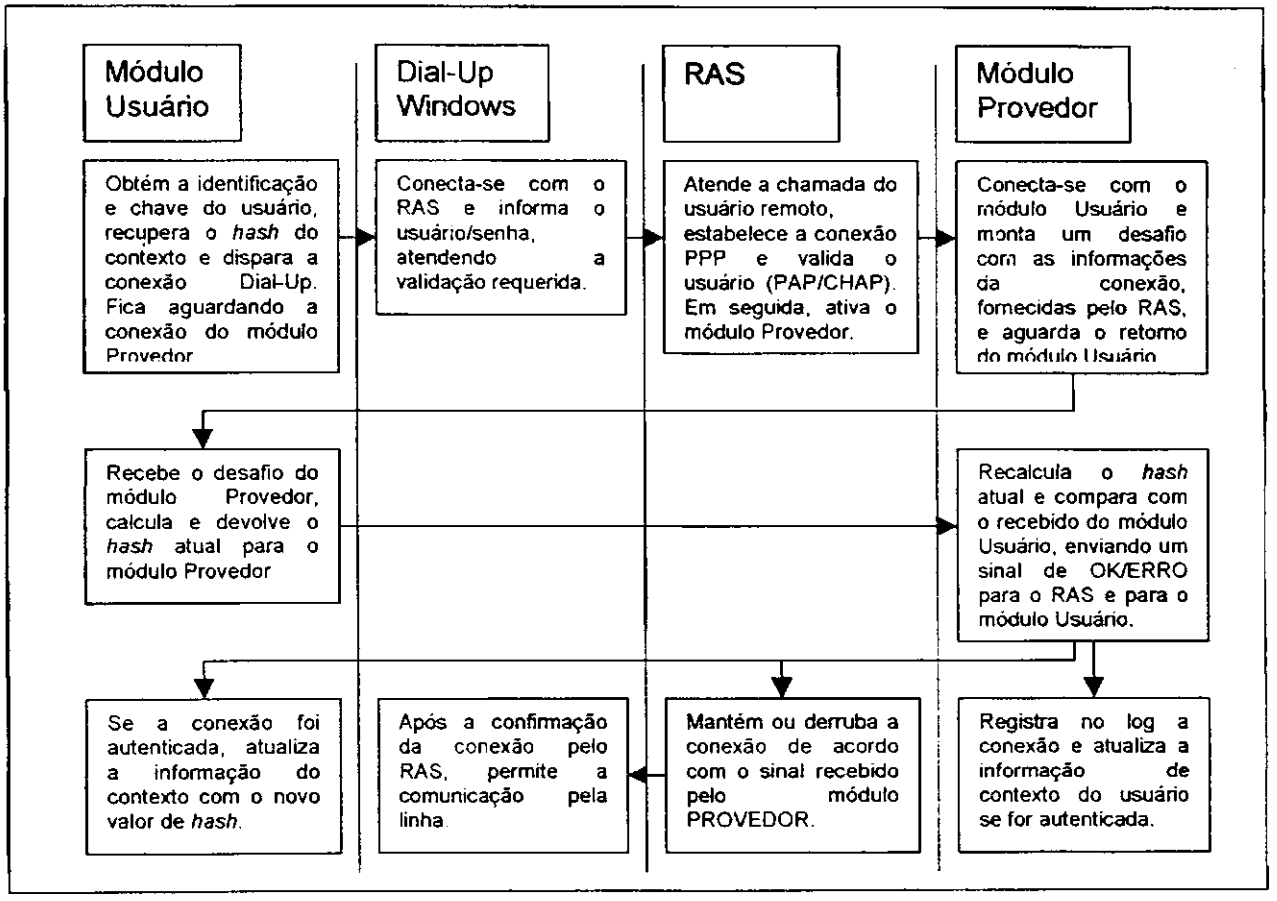

Figura 35: **Fluxo de Autenticação de Acesso Remoto usando Encadeamento**

Existe a possibilidade de perda de sincronismo entre o provedor e o cliente por uma falha na rede, ou seja, em algum momento durante a autenticação a comunicação pode ser perdida e o saldo pode ser que o contexto seja atualizado em uma das partes e na outra não. Isto causará a não validação na conexão na próxima tentativa. Há várias formas de abordar este problema, dentre elas está a montagem de um protocolo mais poderoso a ser implementado entre as aplicações de tal forma que houvesse uma troca de mensagens mais elaborada

antes que os contextos fossem realmente atualizados. Outra forma, talvez mais simples, é dotar o módulo provedor de uma certa inteligência, para em caso de não validação do contexto do módulo usuário, tentar também com o contexto imediatamente anterior. A forma prevista para estes casos na aplicação exemplo aqui proposta é através de uma resincronização manual, após a reclamação do usuário. A técnica *three-way handshake*[31] usada pelo protoloco TCP [COM91] também pode ser aplicada. Neste caso, o módulo USUÁRIO ao receber a mensagem de autenticação da conexão enviada pelo módulo PROVEDOR, alteraria o seu contexto na camada persistente e devolveria uma mensagem de confirmação para o módulo PROVEDOR. Caso o módulo PROVEDOR não recebesse tal confirmação, marcaria esta ocorrência para tentar um sincronismo automático na próxima tentativa de conexão do usuário, pois devido ao erro na conexão o módulo USUÁRIO poderia tanto ter tido tempo de atualizar o seu contexto quanto ainda estar com o contexto anterior. Assim, para manter a segurança o sincronismo automático só seria tentado quando houvesse a indicação na base persistente do módulo PROVEDOR da ocorrência de uma conexão incompleta.

DATA	HORA	PRT	TTY	VEL	IP LOCAL	IP REMOTO	LOGIN	HASH ANT	HASH ATU	RESULTADO
Tue Dec	1 07:48:27 EDT 1998	ppp3	/dev/ttyC7	38400	150.165.249.99	150.165.249.107	rostand	188371375	1422878854	AUTENTICADA
Sat Dec	5 09:33:03 EDT 1998	ppp2	/dev/ttyC0	38400	150.165.249.99	150.165.249.100	rostand	1422878854	1693702256	AUTENTICADA
Thu Dec	10 07:55:33 EDT 1998	ppp0	/dev/ttyC0	38400	150.165.249.99	150.165.249.100	rostand	1693702256	1331292724	AUTENTICADA
Tue Dec	15 07:14:27 EDT 1998	ppp0	/dev/ttyC0	38400	150.165.249.99	150.165.249.100	rostand	1331292724	1603718777	AUTENTICADA
Thu Dec	17 07:13:41 EDT 1998	ppp1	/dev/ttyC7	38400	150.165.249.99	150.165.249.107	rostand	1603718777	14236298	AUTENTICADA
Thu Dec	17 07:21:04 EDT 1998	ppp1	/dev/ttyC4	38400	150.165.249.99	150.165.249.114	rostand	14236298	1143075169	AUTENTICADA
Sun Dec	20 14:03:02 EDT 1998	ppp2	/dev/ttyC9	38400	150.165.249.99	150.165.249.108	rostand	1143075169	1675636184	AUTENTICADA
Mon Dec	21 08:16:44 EDT 1998	ppp0	/dev/ttyC1	38400	150.165.249.99	150.165.249.101	rostand	1675636184	511797366	AUTENTICADA
Tue Dec	22 23:21:43 EDT 1998	ppp1	/dev/ttyC7	38400	150.165.249.99	150.165.249.107	rostand	511797366	1719391233	AUTENTICADA
Thu Dec	24 10:10:45 EDT 1998	ppp1	/dev/ttyC2	38400	150.165.249.99	150.165.249.102	rostand	1331292724	1719391233	RECUSADA

Figura 36: **Exemplo de *log* de encadeamento de conexões com *hashs* explícitos**

[31] Este tipo de protocolo também é usado em aplicações bancárias, como por exemplo saques inter-agência e onde as três mensagens trocadas entre as agências para a realização de uma transação são comumente chamadas de "cruzadas".

Como foi citado anteriormente, a informação do contexto das conexões realizadas por um determinado usuário são mantidas no registro do Windows em formato criptografado. A chave de criptografia é escolhida pelo próprio usuário no momento da primeira conexão, não sendo conhecida nem mesmo pelo provedor de acesso remoto. Se o usuário digitar incorretamente a chave de criptografia do contexto, fatalmente a decriptografia não será realizada corretamente e o valor do contexto seria informado erroneamente para o módulo Provedor. Isto causaria a não autenticação da conexão e, dependendo da estratégia adotada, o bloqueio da conta do cliente. Como tal erro é muito comum, foi criado um pequeno truque para detectar quando o usuário digitou a chave de forma incorreta: uma pequena marca é acrescentada ao valor de *hash* antes de ser criprografado e conferida após a decriptografia ser realizada e antes da conexão ser tentada. Com isto, evitamos uma recusa da conexão pelo módulo PROVEDOR e também criamos uma maior cumplicidade do usuário com o contexto de conexões e menos argumentos para repudiação das mesmas, pois a possibilidade de roubo de contexto fica mais remota. Há um efeito colateral nesta técnica pois, como é avisado ao usuário que ele teclou uma chave incorreta (Figura 37), ataques baseados na força bruta podem ser tentados para identificar o valor do contexto do usuário. Em situações de maior criticidade, esta característica pode ser revista em prol de maior segurança.

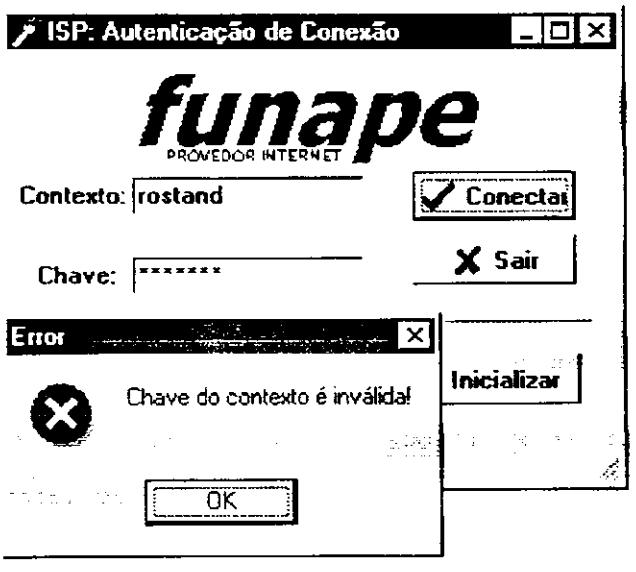

Figura 37: **Erro de Informação da Chave do Contexto**

Foram incluídas na aplicação duas operações especiais de importação e exportação de contexto, como forma de viabilizar o uso da conta de acesso remoto em diversas locações sem perder o contexto. Neste casos, o usuário usa a função de **Exportar**, para transferir o seu contexto do seu microcomputador para um disquete. No microcomputador destino, ele usa a função **Importar** para transferir o contexto do disquete para o registro do Windows. O

processo pode ser realizado sempre que necessário. Também é possível manter, no mesmo microcomputador, informações sobre o contexto de vários usuários. Os dados mantém-se criptografados durante a importação e exportação.

A aplicação aqui discutida foi implementada, e está em funcionamento em pequena escala, no provedor de acesso à Internet mantido pela **FUNAPE – Fundação de Apoio à Pesquisa e à Extensão** da **UFPB - Universidade Federal da Paraíba – Campus I** e operado pelo **NTI – Núcleo de Tecnologia da Informação** da mesma Instituição. Todas as conexões para acesso à conta pessoal do autor situada neste provedor no período de março de 1998 até agora, foram realizadas usando esta aplicação. Os resultados foram muito bons, com quase que total transparência com relação a não usar o mecanismo adicional de validação e também sem que ocorresse nenhum falsa negativa de acesso, ou seja, a característica de disponibilidade do serviço não foi prejudicada. Após solicitação do autor, algumas pessoas se dispuseram a tentar quebrar o esquema de segurança, fazendo conexões ilegítimas sem usar a aplicação exemplo. Para facilitar o trabalho dos colaboradores, o autor forneceu a senha real de acesso à sua conta no provedor, mas, apesar disto, não houve a quebra do encadeamento, apenas o registro no *log* das tentativas de conexão infrutíferas. É uma aplicação perfeitamente funcional, contando inclusive com um módulo de testes (Figura 38) e adaptações futuras podem ser feitas com facilidade. Além disso, o sentimento do autor é que tais modificações sejam de pequena monta e fiquem restritas apenas à integração com o servidor de autenticação utilizado (RADIUS ou TACACS), pois os módulos USUÁRIO e PROVEDOR são bastante estáveis.

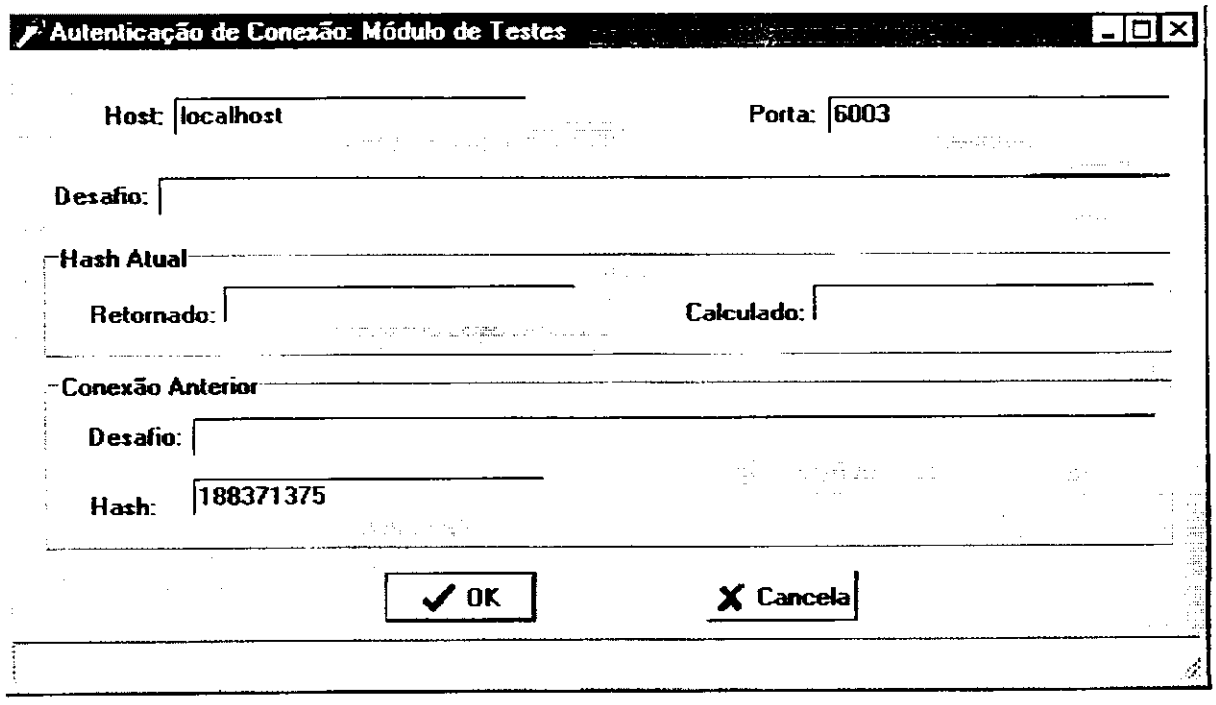

Figura 38: **Módulo de Testes de Autenticação de Conexões**

5.12. Sumário

Neste capítulo foi apresentado o encadeamento de transações como uma forma acessória de incrementar a segurança e a confiabilidade de aplicações distribuídas em geral, através da criação de um controle fim-a-fim. A idéia principal é que, a partir de elementos simples como algoritmos de *secure hash*, *triggers* de banco de dados, *stored procedures* e algumas modificações nos procedimentos, podemos criar um encadeamento de transações entre os lados cliente e servidor. Isto cria a idéia de um relacionamento exclusivo, de uma contextualização específica entre um determinado **comprador** e um determinado **vendedor**, por exemplo. Desta forma, qualquer desvio deste contexto, poderia ser detectado tanto durante a realização da transação quanto posteriormente, através de procedimentos *batch*. O problema de autenticação de usuários de provedores de acesso remoto, que envolve um ambiente distribuído e está associado com a Internet, é apresentado para exemplificar o uso da técnica do encadeamento de transações. Usando o encadeamento de transações, passamos a encadear conexões e o principal fator para o incremento da segurança é que, ao invés de contarmos apenas com uma autenticação baseada em identificação/senha, passamos a contar também com uma espécie de *one-time password* lógico, solicitado como desafio pelo provedor. Este valor, que é o *hash* do encadeamento das conexões anteriores, mudará constantemente e será válido apenas para uma determinada conexão. A aplicação resultante é simples, voltada quase que exclusivamente para a autenticação de usuários, e possui dois componentes distribuídos, no modelo cliente/servidor: um servidor de validação de contexto (módulo USUÁRIO) e um cliente desafiador de contexto (módulo PROVEDOR). No próximo capítulo, faremos uma análise final dos assuntos abordados e dos resultados obtidos.

6. Considerações Finais

Atuando principalmente no aspecto de detecção e prevenção de fraudes, que envolve a característica de integridade, o encadeamento de transações pode ser visto como um excelente parceiro para as outras técnicas de segurança que atuam com mais eficiência no aspecto da disponibilidade e confidencialidade da informação. Um dos principais motivos que nos leva a esta conclusão é a aderência da técnica de encadeamento aos três princípios da segurança computacional, quais sejam: *Princípio da Penetração mais Fácil*, *Princípio da Proteção Adequada* e *Princípio da Eficácia* [BIR96].

A proteção posterior fornecida às bases de dados permanentes do servidor, o receptáculo final das informações, oferece um mecanismo de proteção que está de acordo com o Princípio da Penetração mais Fácil, que prega que um intruso vai usar qualquer meio disponível de penetração que, necessariamente, não é o mais óbvio e muito menos o mais protegido.

> "Hoje em dia, a maneira mais fácil de danificar ou agir de forma fraudulenta seria acessar os bancos de dados relativamente desprotegidos e alterar os dados." [WHI97]

O importante papel que as senhas e outras informações sigilosas possuem no processo de autenticação exige um grande esforço na sua guarda, pois elas devem ser, constantemente, mantidas fora do alcance de intrusos para que a segurança seja garantida. Pelo Princípio da Proteção Adequada, que reza que bens computacionais devem ser protegidos somente até eles perderem o seu valor, o encadeamento de transações proporciona uma diminuição do valor potencial da senha, ao associar a ela, outras informações de caráter dinâmico, com usabilidade limitada pelo tempo. A senha sozinha não possui tanto valor e a informação de contexto envelhece, o que diminui a potencialidade do seu uso por muito tempo.

O último princípio, o da Eficácia, diz que os controles de segurança devem ser eficientes, fáceis de serem usados e adequados. Podemos facilmente enquadrar o encadeamento por este princípio, talvez uma das suas características mais fortes, pois ele é simples e totalmente flexível no uso de ferramentas para a sua implantação, o que facilita a sua adequação a diversos ambientes, inclusive em aplicações legadas. Na construção ou adequação de uma aplicação para o uso do encadeamento de transações, pode ser usada qualquer combinação de técnicas, protocolos ou ferramentas disponíveis para criptografia, armazenamento de dados, *secure hashing*, protocolo de comunicação etc sem

descaracterizar o algoritmo. Assim, podem ser usadas ferramentas simples para aplicações com menores requisitos de segurança enquanto que aplicações mais críticas podem ser construídas com mecanismos de maior complexidade.

Diversas melhorias podem ser acrescidas ao modelo aqui proposto, como por exemplo o aumento de informações armazenadas sobre o contexto e o desenvolvimento de mecanismos mais elaborado que permita que, senão todas, pelo menos parte das exceções detectadas pela quebra do encadeamento possam ser manipuladas automaticamente. Ainda como melhoramento futuro, pode ser discutido mais detalhadamente a viabilidade de uso do encadeamento de transações em aplicações voltadas para a Internet e orientadas a conexões com servidores *stateless*[32]. Nestes casos, é necessário dotar a aplicação cliente de um mecanismo que permita aos módulos clientes manter um mínimo de informações persistentes a respeito do seu relacionamento com um determinado módulo servidor. Uma alternativa a ser explorada é o uso de HTTP *cookies* (*Persistent Client State*) [GS97].

A própria aplicação exemplo, mostrada no Capítulo 7, representa uma demanda do mercado de provedores de acesso remoto e também merece um desdobramento. Durante a sua elaboração, foram entrevistados alguns responsáveis por provedores comerciais em João Pessoa, entre eles a Openline Internet Provider e a Netway BBS, os maiores do mercado, com penetração inclusive em outras cidades do Estado. A posição foi unânime: realmente existe um problema sério de autenticação de seus usuários, principalmente pela dificuldade de conscientização para o uso de senhas adequadas. Entre as melhorias possíveis na aplicação exemplo, está um melhor acabamento no aspecto de instalação e documentação e também a sua adaptação para os protocolos RADIUS e TACACS, considerando-se a possibilidade de usar a própria base de dados mantida por estes protocolos para bilhetagem para armazenar também o contexto dos usuários.

Uma limitação do encadeamento de transações é que o mesmo só se aplica para aquelas aplicações onde a interação entre o cliente e o servidor se dá através de um canal de comunicação FIFO (*First In First Out*), ou seja, as transações devem ser processadas pelo servidor na mesma ordem em que são geradas pelo cliente. Este entretanto, é o modelo seguido por um grande número de aplicações distribuídas.

Uma outra limitação é que por basear-se na criação de contextos entre os lados cliente e servidor, que devem permanecer sincronizados, o encadeamento de transações não se aplica em situações em que o anonimato é desejado, como em algumas aplicações voltadas

---

[32] Servidores stateless não mantém informações sobre o estado de sessões [BIR96].

para a disponibilização de dinheiro verdadeiramente eletrônico ou digital [OO92]. Entretanto, aplicações com esta característica têm pouquíssima ou nenhuma demanda atualmente, pois mesmo as aplicações voltadas para comércio eletrônico têm sido idealizadas *como "front-end"* para empresas de administração de cartões de créditos ou bancos, que baseiam-se no conhecimento das transações realizadas por seus clientes.

Acreditamos que o mecanismo de encadeamento de transações para controle fim-a-fim, por sua simplicidade, pode ser implementado com facilidade em muitos casos, quer seja como forma adicional de controle em situações que tenham a segurança como fator crítico, apoiando os outros mecanismos de proteção já implementados, quer seja como única forma de validação em aplicações com menores requisitos de segurança.

7. Referências Bibliográficas

[ABF92] E.Armstrong, S.Bobrowski, J. Frazzini. *ORACLE7 Server Applications Developer's Guide*. Oracle Corporation. Redwood City, CA 1992.

[BAU98] D. Baum. *Fast Track to E-Commerce - All Aboard to Eletronic Commerce Train*. Em Oracle Magazine. Oracle Publishing. Redwood Sores, Janeiro/ 1998.

[BLU94] C. Blum. *The Jargoon File 3.0.0*. [Online]. Disponível: http://www.klammeraffe.org/jargon/logic_bomb.html. [1999, Fevereiro, 28]

[BIR96] K. P. Birman. *Building Secure and Reliable Network Applications*. Manning Publications Co. Greenwich, 1996.

[BRO98] B. D. Brown. *Securing Transactions in Network Applications*. Oracle Magazine. Oracle Corporation. Redwood City - CA, Janeiro/1998.

[CAS97] F. Castejon. *Dinheiro Eletrônico: Visão de "Porta-Moedas"*. CIAB 97. São Paulo, Junho/1997.

[CIS95] Cisco Systems Inc. *Single-User Network Access Security TACACS+*. Cisco White Papers. 1996 [Online]. Disponível: http://www.cisco.com/warp/public/614/7.html. [1999, Fevereiro, 28].

[COM91] D. E. Comer. *Internetworking with TCP/IP – Volume I*. Prentice-Hall International. New Jersey, 1991.

[CSI98] Computer Security Institute. *The Cost of Computer Crime*. CSI/FBS Computer Crime Survey, 1998. [Online]. Disponível: http://www.gocsi.com/losses.htm. [1999, Fevereiro, 28].

[CS91] D. E. Comer, D. L. Stevens. *Internetworking with TCP/IP – Volume II*. Prentice-Hall International. New Jersey, 1991.

[DAT91] C. J. Date; *Introdução a Sistemas de Bancos de Dados*. Editora Campus. Rio de Janeiro, 1991.

[DW92] J. Duntemann, K. Weiskamp. *C/C++ Ferramentas Poderosas*. Berkeley Brasil Editora. Rio de Janeiro, 1992.

[EFL97] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, & N. R.

Mead. *Survivable Network Systems: An Emerging Discipline*. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Novembro/1997.

[FAB97] M. Fabiani. *Hackers*. Revista Internet World. Mantelmedia Editora. Rio de Janeiro, Julho/1997.

[FRA97] R. Franz. *Tendências nos Canais Alternativos*. CIAB 97. São Paulo, Junho/1997.

[FRI98] Aharon Friedman. *Virtual Private Networks*. CIAB 98. São Paulo, 1998.

[GAM86] W. F. Giozza. J. F. M. Araújo. J. A. B. Moura, J. P. Sauvé. *Redes Locais de Computadores*. McGraw Hill, São Paulo/1986.

[GEE98] D. E. Geer. *Risk Management is where the money is*. Digital Commerce Society of Boston. Cambridge, Novembro/1998.

[GS96] S. Garfinkel, G. Spafford. *Pratical Unix & Commerce*. O´Reilly Associates, Inc. Cambridge, 1996.

[GS97] S. Garfinkel, G. Spafford. *Web Security & Commerce*. O´Reilly Associates, Inc. Cambridge, 1997.

[GUR97] H. Gurovitz. *Lute com as mesmas armas do inimigo*. Revista Exame. Editora Abril. Rio de Janeiro, Novembro/1997.

[IW97] Information Week Magazine. *Information Week/Ernst Young Security Survey*. Setembro, 1997. [Online]. Disponível: http://www.informationweek.com/647/security.htm. [1999, Fevereiro 28].

[HIC95] K. E. B. Hickman. *The SSL Protocol*. Netscape Communications Corp. Janeiro/1995.

[HIG97] M. Higgins. *Segurança na Internet*. CIAB 97. São Paulo, Junho/1997.

[HIL97] M. Hill. *Chaves tornam o comércio seguro*. Revista LAN Times. Editora Rever. São Paulo, Novembro/1997.

[HUN94] C. Hunt. TCP/IP *Network Administration*. O'Reilly & Associates, Inc. Sebastopol, 1994.

[HUR97] J. Hurwitz. *Segurança Distribuída - Uma abordagem para encontrar e solucionar falhas de segurança*. Em DBMS. MANtelMedia Editora. Rio de

Janeiro, Dezembro/1997.

[HY95] T. J. Hudson, E. A. Young. *SSL Programmer Reference*. Mincom Pty Ltd. Julho/1995.

[KIN97] C. M. King. Public Key Infrastructure: End to End Security. Infosec Engineering. Chelmsford, MA 1997.

[KPS95] C. Kaufman, R. Perlman e M. Speciner. *Network Security.* Prentice Hall. New Jersey, 1995.

[LC97] Li, X. & Crane, N.B. (1997, October 29). *Bibliographic Formats for Citing Electronic Information*, [Online]. Disponível: http://www.uvm.edu/~ncrane/estyles/ [1999, Fevereiro 28].

[LEE96] J. A. N. Lee. *Hacking*. MacMillan Encyclopedia of Computers. Outubro/1996.

[LUC97] Lucent Technologies, Inc. *RADIUS(TM) Administrator's Guide*. Maio, 1997. [Online] Disponível: http://www.livingston.com/tech/docs/radius/Title.fm.html [1999, Fevereiro 28].

[LOB99] M. Lobel. *The Case for Strong User Authentication.* Security Dynamics Technologies, Inc. [Online] Disponível: http://www.securitydynamics.com/products/whitepapers/casestrong-wp.html#userstrong [1999, Fevereiro 28].

[MB98] C.C Monteiro, F.V. Brasileiro. *Groove: um Ambiente para Auxiliar a Manutenção de Segurança em Sites Internet*. Anais do IX Congresso Latino-Iberoamericana de Investigation Operativa. 1998.

[MER98] A. M. Mercês. *Auditoria de Informática*. Audibra. Recife, Março/1998.

[MGF97] A. L. Moreira, D, H. Goya, E. T. Freitas. *Você está seguro?*. Revista PC Magazine Brasil. Editorial América do Brasil. São Paulo, Dezembro/1997.

[NER98] F. Nery. *Tendências Mundiais em Segurança para Internet e Intranet*. CIAB 98. São Paulo, 1998.

[OLI94] P. B. de Oliveira. *Manual do Auditor Interno*. Instituto dos Auditores do Brasil. São Paulo, 1994.

[OO92] T. Okamoto e K. Otha. *Universal eletronic cash*. Em Advances in Cryptology-CRYPTO'91 Proceedings. Springer-Verlag, 1992.

[PAV98] L. Pavani. *Mas ... você é você mesmo?*. Revista INFO EXAME. Editora Abril. Rio de Janeiro, Maio/1998.

[PFL97] C. P. Pfleeger. Security in Computing. Prentice Hall. New Jersey, 1997.

[PSI98] Módulo Consultoria em Informática Ltda. *4ª Pesquisa Nacional sobre Segurança da Informação*. Security Forum'98 - 2º Congresso Nacional sobre Segurança e Auditoria da Informação, Setembro/1997 [Online]. Disponível: http://www.modulo.com.br [1999, Fevereiro 28].

[RID97] L. Ridolfi. *Técnicas de Testes de Vulnerabilidade*. CIAB 97. São Paulo, Junho/1997.

[RUG97] W. Ruggiero. *Segurança na Internet e Comércio Eletrônico*. CIAB 97. São Paulo, Junho/1997.

[SAN97] T. W. Sandholm. *Unenforced E-Commerce Transactions*. IEEE Internet Computing. IEEE Computer Society Publications. Los Alamitos – CA, Dezembro/1997.

[SCH91] H. Schildt. C++ - *The Complete Reference*. Editora McGraw Hill, São Paulo/1991.

[SDT99] Security Dynamics Technologies, Inc. *Understanding Public Key Infrastructure (PKI)*. [Online] Disponível: http://www.securitydynamics.com/products/whitepapers/pki/index.html [1999, Fevereiro 28]

[SIM92] W. Simpson. *The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links*. Network Working Group. RFC 1331, Maio/1992.

[TLA98] Telecom and Logistics Associates. *Computer Crime and Security Survey*. Results of the FBI/CSI 1998. [Online] Disponível: http://www.tla.ch/TLA/NEWS/f3survey98.htm [1999, Fevereiro 28].

[WAY97] P. Wayner. *Digital Cash - Commerce on the Net*. Academic Press Limited. London, 1997.

[WHI97] D. White. *Mais segurança nos Sistemas Distribuídos*. DBMS Tools & Strategies for IS Professionals. MantelMedia Editora. Rio de Janeiro, Dezembro/1997.

[WSJ96a] *Artigo sobre "pachinko losses"*. Wall Street Journal, 26 de Maio de 1996.

[WSJ96b] *Artigo sobre "pachinko losses"*. Wall Street Journal, 24 de Julho de 1996.

ANEXO A: SISTEMAS CRIPTOGRÁFICOS

A encriptação ou criptografia já estava presente no sistema de escrita hieroglífica dos egípcios. Desde então vem sendo muito utilizada, principalmente para fins militares e diplomáticos. No âmbito da computação, a criptologia é um dos mecanismos de segurança mais utilizados, principalmente quando há a necessidade de se manter o sigilo das informações manipuladas. Também é muito usada para se codificar dados e mensagens antes que estes sejam enviados por vias de comunicação, para que, mesmo que sejam interceptados, não possam ser interpretados.

Dentre as propriedades associadas a criptografia computacional, destacam-se:

- **Sigilo**: restrição do acesso à informação;
- **Integridade da Informação**: garantia de que a informação não foi alterada de forma intencional ou acidental;
- **Autenticação em comunicações**: possibilidade de verificação e autenticação mútua das partes envolvidas em uma comunicação;
- **Autenticação do Remetente**: garantia para o destinatário de uma mensagem de que a mesma foi realmente enviada pelo remetente esperado;
- **Autenticação do Destinatário**: garantia para o remetente de uma mensagem de que a mesma só poderá ser acessada pelo destinatário legítimo.

A.1. Segurança de Sistemas Criptográficos

A segurança de um sistema criptográfico não pode estar baseada nos algoritmos de codificação e decodificação adotados mas em um valor: a chave utilizada. O mecanismo deve ser tão seguro que nem mesmo o autor de um algoritmo deve ser capaz de decodificar uma mensagem se não possuir a chave. Pela premissa de Kerckhoffs, matemático holandês do século XIX, assume-se que todo o mecanismo criptográfico pode ser público, exceto as chaves. Para um algoritmo criptográfico ser considerado seguro, deve atender às seguintes premissas:

- O criptoanalista[33] tem acesso à descrição completa do algoritmo;
- O criptoanalista tem acesso a grandes volumes de mensagens originais e suas correspondentes mensagens cifradas;
- O criptoanalista é capaz de escolher quais mensagens devem ser cifradas e receber as mensagens cifradas correspondentes.

Um atacante passivo somente acessa os elementos (**interceptação**) enquanto que um atacante ativo pode alterar alguns destes elementos (**modificação**). Os tipos de ataque (ou **criptoanálise**) mais comuns são:

- **ataque do texto cifrado** (*cyphertext-only*): o criptoanalista tem a sua disposição uma grande quantidade de mensagens cifradas, mas desconhece as originais e as chaves utilizadas. A sua tarefa é recuperar as mensagens originais;
- **ataque do texto conhecido** (*known-plaintext*): o criptoanalista tem a sua disposição uma grande quantidade de mensagens cifradas e também as mensagens originais equivalentes. Sua tarefa é deduzir as chaves usadas;
- **ataque adaptativo do texto escolhido** (*adaptative-choosen-plaintext*): neste ataque, ele tem a possibilidade de fornecer um pequeno conjunto de mensagens para serem cifradas e analisar os resultados obtidos, fornecer outro conjunto e assim por diante. Para deduzir as chaves utilizadas, o criptoanalista dispõe de uma "caixa-preta" de ciframento;
- **ataque do texto cifrado escolhido** (*choosen-cyphertext*): o criptoanalista, além de uma grande quantidade de mensagens e seus equivalentes cifrados, pode produzir uma mensagem cifrada específica para ser decifrada e analisar o resultado produzido. Neste caso, o criptoanalista possui uma "caixa-preta" de deciframento para deduzir as chaves utilizadas.

Um sistema é dito seguro se ele é teoricamente inquebrável, ou seja, não interessa a quantidade de texto normal ou decifrado a disposição, nunca se tem informação suficiente para deduzir as chaves utilizadas ou decifrar um texto qualquer cifrado. Só se conhece um método nesta categoria: a Cifra de Vernam ou *one-time pad* (cifra de uso único). Ele consiste na premissa de que dois elementos que desejam se comunicar possuem cópias

---

[33] Uma pessoa não autorizada que tem acesso a alguns dos elementos de um criptosistema é denominado atacante ou **criptoanalista**.

idênticas de uma seqüência randômica de valores, que são usados como chave. O método, entretanto, exige que cada chave seja usada uma única vez e que o comprimento da seqüência (chave) seja maior ou, no mínimo, igual ao comprimento da mensagem a ser cifrada.

A.2. Métodos de Cifragem

A.2.1. Cifragem por Substituição

Este tipo de criptografia consiste em trocar cada caracter ou grupo de caracteres por outro, de acordo com uma tabela de substituição. Pode-se quebrar este método analisando-se a freqüência de cada caracter no texto cifrado e comparando-se estas freqüências com aquelas que normalmente aparecem em um determinado idioma. As vogais têm maior freqüência do que as consoantes e alguns caracteres possuem freqüência muito baixa em relação aos demais. Para amenizar a freqüência de caracteres, podemos utilizar várias tabelas para a cifragem de um texto. Para uma substituição mono-alfabética, podemos ter N! tabelas de substituição, onde **N** é a quantidade de símbolos do alfabeto utilizado. Deve existir uma chave que indica qual das tabelas será usada para cada letra do texto original. Desta forma, quanto maior a chave mais seguro é o método. Entretanto, já é possível realizar criptoanálises eficazes com a simples descoberta do tamanho da chave **k** e analisando blocos de **k** caracteres no texto cifrado, verificando a frequência de repetição dos caracteres.

- **Substituição Mono-Alfabética**: cada letra do texto original é trocada por outra de acordo com uma tabela e com sua posição no texto. A **Substituição de César** é um exemplo de substituição mono-alfabética que consiste em trocar cada letra por outra que está **n** letras adiante na ordem alfabética, onde **n** é a chave do ciframento. Por exemplo, se **n** é 3 a letra **a** é trocada pela letra **d**, e assim por diante. Como existem apenas **k** chaves possíveis, onde **k** é a quantidade de símbolos do alfabeto, não é um método muito seguro, pois é extremamente fácil decifrá-lo pelo método da força bruta;
- **Substituição por Deslocamento:** a chave indica quantas posições devem ser avançadas no alfabeto para substituir cada letra. De forma diferente da substituição de César, as letras não são trocadas sempre por uma letra **n** posições à frente no alfabeto. Considerando a chave **020813**, a primeira letra será trocada pela letra que está duas posições a frente no alfabeto, a segunda pela que está 8 posições a frente no alfabeto e assim por diante, repetindo a seqüência da chave até o final da mensagem (se for necessário. Por exemplo, a seqüência **UFPB** seria cifrada para **WNCD**;

- **Substituição Monofônica:** como a anterior, mas agora cada caracter pode ser mapeado para um ou vários caracteres na mensagem cifrada. Isto evita a linearidade da substituição;
- **Substituição Poli-Alfabética:** combinação do uso de várias substituições mono-alfabéticas, usadas em rotação de acordo com um critério ou chave. Por exemplo, poderiam ser utilizadas 5 tabelas, usadas em alternância a cada 5 caracteres;
- **Substituição por Polígramos:** utiliza um grupo de caracteres ao invés de um caracter individual. Se fossem considerados trigramas, por exemplo, **UVA** poderia ser substituído por **XTS** ou **KQR**.

A.2.2. Cifragem por Transposição

Troca-se a posição dos caracteres na mensagem. Por exemplo, o texto pode ser reescrito percorrendo-o por colunas. Pode-se ainda definir um tamanho para um vetor de trocas e também a ordem em que as trocas serão feitas. A chave pode ser usada para definir isto. Por exemplo, em um vetor de tamanho 6, podemos trocar o primeiro caracter pelo terceiro, o segundo pelo quinto e o quarto pelo sexto.

Se a freqüência dos caracteres for a mesma do idioma, usamos cifragem por transposição. Se for diferente, usamos a cifragem por substituição. Também é possível combinar substituição com transposição e vice-versa.

A.2.3. Cifragem de Blocos

Os métodos de cifragem de blocos ao invés de se basearem em palavras ou caracteres se baseiam em blocos de bits. Existem vários tipos de métodos que realizam cifragem sobre blocos, os principais são:

- **Método do Livro de Códigos** (*electronic code book* – ECB): Cada bloco da mensagem original é individual e independentemente cifrado para produzir os blocos da mensagem cifrada. O bloco típico tem 64 bits, o que produz um livro de códigos de 2^{64} entradas, além disso, para cada chave possível existe um livro de códigos diferente. A vantagem do método é a sua simplicidade e a independência entre os blocos. A desvantagem é que um criptoanalista pode começar a compilar um livro de códigos, mesmo sem conhecer a chave. Um problema mais grave é a chamada repetição de bloco, onde um atacante ativo pode alterar parte de uma mensagem criptografada sem saber a chave e nem mesmo o conteúdo que foi modificado. É possível interceptar uma transação bancária de transferência de saldo entre contas de duas pessoas quaisquer. Em

seguida, pode ser feita a interceptação de uma transação legítima de transferência de saldo para a conta do atacante. Com tais elementos a mão, é possível identificar os blocos correspondentes ao destinatário e dessa forma substituir o destinatário em todas as mensagens seguintes;

- **Método de Encadeamento de Blocos** (*cipher block chaining* – CBC): O CBC realimenta a cifragem do bloco atual com o resultado da cifragem dos blocos anteriores. A operação mais utilizada é o ou-exclusivo com o bloco anterior. Desta forma, os blocos iguais serão normalmente cifrados de forma diferente, desde que, no mínimo, um dos blocos anteriores da mensagem seja diferente. Entretanto, duas mensagens iguais serão mapeadas para os mesmos blocos e mensagens com início igual serão cifradas da mesma forma até que ocorra a diferença. A maneira de se evitar este problema é a utilização de um vetor de inicialização distinto para cada mensagem;
- **Método da Realimentação de Cifra** (*cipher feedback* – CFB): Quando há necessidade de se enviar mensagens que possuem tamanho menor do que um bloco, usa-se o método CFB, que trabalha com grupos ou sub-blocos (8 bits, por exemplo). Neste caso, a realimentação é feita sobre o grupo, também utilizando-se o ou-exclusivo;
- **Método de Encadeamento de Blocos** (*block chaining* - BC): a entrada do cifrador é operada com um ou-exclusivo do bloco com o ou-exclusivo de todos os blocos anteriormente cifrados;
- **Método de Encadeamento Propagado** (*propagating cipher block chaining* – PCBC): a entrada do cifrador é alimentada com um ou-exclusivo com o ou-exclusivo de todos os blocos normais e cifrados anteriores.

A.3. Sistemas Criptográficos Seguros

Quando o sistema criptográfico utiliza para o deciframento a mesma chave usada para o ciframento (ou que possa ser deduzida em função desta), classifica-se como um **sistema criptográfico de chaves simétricas** (ou **chave secreta**). Caso as chaves sejam diferentes e não dedutíveis uma da outra, o sistema é classificado como **sistema criptográfico de chaves assimétricas** (ou **chave pública**). Os algoritmos simétricos são executados mais rapidamente do que os assimétricos, que, por sua vez, são mais robustos. Algoritmos modernos de criptografia não podem ser computados por humanos. Os mais fortes e poderosos são desenvolvidos para serem executados por computadores ou dispositivos especiais de *hardware*.

A.3.1. Sistemas de Chave Única ou Secreta (Simétricos)

O exemplo mais difundido de cifrador computacional de chave única é o **DES** (*Data Encryption Standard*), desenvolvido pela IBM e adotado como padrão nos EUA em 1977. O DES cifra blocos de 64 bits (8 caracteres) usando uma chave de 56 bits com 8 bits de paridade. O algoritmo inicia realizando uma transposição inicial sobre os 64 bits da mensagem, seguida de 16 passos de cifragem e conclui realizando uma transposição final, que é o inverso da transposição inicial. Para os 16 passos de cifragem usam-se 16 sub-chaves, todas derivadas da chave original através de deslocamentos e transposições.

Os passos de cifragem do DES têm dois objetivos básicos: a **difusão** e a **confusão**. A **difusão** visa eliminar a redundância existente na mensagem original, distribuindo-a pela mensagem cifrada. O propósito da **confusão** é tornar a relação entre a mensagem e a chave tão complexa quanto possível. O DES pode ser quebrado pelo método da força bruta, que consiste em tentar todas as combinações possíveis de chave. Como a chave tem 56 bits, existe um total de 2^{56} chaves possíveis.

Existem diversos algoritmos de cifragem de blocos de chave única, como:

- **IDEA**: blocos de 64 bits com chave de 128 bits;
- **Triple-DES**: o DES é aplicado três vezes, com sequências de cifragem e decifragem, combinando a utilização de 2 chaves;
- **NewDES**: blocos de 64 bits e chave de 120 bits;
- **Khufu e Khafre**: trabalham de forma semelhante ao DES, mas usam tabelas de substituição de 256 posições de 32 bits -- contra as de 6 posições de 4 bits do DES. Trabalham com chaves de 512 bits e permitem um número de passos flexível, múltiplo de 8;
- **Lucifer**: precursor do DES;
- **Madryga**: trabalha com 8 bits, usando ou-exclusivo e deslocamento de bits;
- **FEAL-N**: baseado no DES, é possível especificar o número de passos da cifragem;
- **REDOC II e III**: realizam operações sobre bytes;
- **LOKI**: bloco e chave de 64 bits;
- **MMB**: blocos e chave de 128 bits;
- **Skipjack**: chave de 80 bits e 32 passos.

A.3.2. Sistemas de Chave Pública (Assimétricos)

Nestes sistemas, a chave de ciframento é publicada ou tornada acessível aos usuários, sem que haja quebra na segurança. Dessa forma, cada usuário tem uma chave de ciframento, de conhecimento público, e outra de deciframento, secreta. Se um usuário **A** deseja mandar uma mensagem para um usuário **B**, ele utiliza a chave de ciframento pública $\mathbf{Pub_B}$ e envia a mensagem para **B**. Este, de posse de sua chave de deciframento secreta $\mathbf{Sec_B}$, decodifica a mensagem.

Um exemplo de um sistema de chave pública é o **RSA**, cujo nome vem das iniciais dos seus criadores: Rivest, Shamir e Adleman. Sua segurança baseia-se na intratabilidade da fatoração do produto de dois primos.

A.4. Assinatura Digital

Nos sistemas com chave pública, qualquer pessoa pode cifrar uma mensagem, mas somente o destinatário desta mensagem pode decifrá-la. Invertendo-se o uso das chaves, podemos ter uma mensagem que só pode ser cifrada por uma pessoa mas decifrada por qualquer um, obtendo-se assim um efeito de personalização da mensagem, semelhante a uma assinatura. Um sistema com tais características é chamado de sistema de assinatura digital.

Para "assinar" uma mensagem, um determinado usuário **A** codifica uma mensagem utilizando a sua chave secreta $\mathbf{Sec_A}$ de tal forma que somente a sua chave pública $\mathbf{Pub_A}$ permitirá a decodificação da mensagem. Isto é a prova que **A**, e apenas **A**, codificou a mensagem. Qualquer um de posse da chave pública $\mathbf{Pub_A}$ de **A** poderá decodificar a mensagem. Para garantir o sigilo da mensagem a ser enviada para um destinatário **B**, a mensagem já criptografada deve ser criptografada novamente, desta vez usando-se a chave pública do destinatário $\mathbf{Pub_B}$, para que somente este possa decodificar a mensagem usando a sua chave secreta $\mathbf{Sec_B}$. Assim, para A assinar uma mensagem para B deve fazer o seguinte:

msg_cifrada = CRYPTO($\mathbf{Pub_B}$, CRYPTO($\mathbf{Sec_A}$, msg_natural))

e B, para recuperar uma mensagem de A, deve realizar os seguintes passos:

msg_natural = CRYPTO($\mathbf{Pub_A}$, CRYPTO($\mathbf{Sec_B}$, msg_cifrada))

A assinatura digital tem as seguintes propriedades:

- **Autoria Garantida:** quando uma mensagem é decifrada com a chave pública $\mathbf{Pub_A}$ de A, confirma-se que foi **A** e somente **A** codificou a mensagem;
- **Autenticidade:** a assinatura não pode ser forjada, pois somente A conhece a sua chave secreta;
- **Integridade**: a mensagem assinada não pode ser alterada, pois se houver qualquer alteração no texto criptografado ele não poderá ser restaurado com o uso da chave pública $\mathbf{Pub_A}$ de **A**;
- **Personalização**: a assinatura não é reutilizável, pois é uma função de uma determinada mensagem e não pode ser transferida para outra mensagem;
- **Não Repudiação**: a assinatura não pode ser repudiada, pois o usuário **B** não precisa de nenhuma ajuda de **A** para reconhecer a sua assinatura e **A** não pode negar ter assinado a mensagem.