EDUARDO PAULO GASPARETTO

SISTEMAS DE ARQUIVOS CRIPTOGRAFADOS

Monografia apresentada ao Departamento de Ciência da Computação DA Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação *Lato Sensu* para a obtenção do título de Especialista em Administração em Redes Linux

Orientador
Prof. Joaquim Quinteiro Uchôa

LAVRAS
MINAS GERAIS - BRASIL
2005

EDUARDO PAULO GASPARETTO

SISTEMAS DE ARQUIVOS CRIPTOGRAFADOS

Monografia apresentada ao Departamento de Ciência da Computação DA Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação *Lato Sensu* para a obtenção do título de Especialista em Administração em Redes Linux

APROVADA em 9 de dezembro de 2006

Prof. Heitor Augustus Xavier Costa

Prof. Herlon Ayres Camargo

Prof Joaquim Quinteiro Uchoa

_____

UFLA
(Orientador)

LAVRAS
MINAS GERAIS - BRASIL

A meus familiares, amigos, professores e companheiros da comunidade do software livre.

AGRADECIMENTOS

Gostaria de agradecer a minha família pelo apoio durante todo o curso, motivando-me quando o desânimo tomava conta.

Também agradeço ao professor Joaquim Quinteiro Uchôa, que sempre esteve disposto a ajudar, e em seu papel de orientador sempre foi muito profissional e motivador.

RESUMO

O objetivo desta monografia é a instalação e a configuração de um sistemas de arquivos criptografado, utilizando ferramentas livres que englobam tanto uma partição cifrada como um arquivo de bloco cifrado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Desde os primórdios da humanidade as pessoas enviam mensagens e, muitas vezes, o autor dessas mensagens não deseja que seu conteúdo seja descoberto por ninguém, a não ser o destinatário. Para proteger o conteúdo de mensagens e de informações que devem permanecer confidenciais, foi criada uma técnica chamada *criptografia*, que significa ‘escrita desconhecida’.

A criptografia evoluiu muito desde o tempo do imperador Júlio César, na Roma Antiga, quando se faziam simples substituições de caracteres por outros caracteres. Hoje, com o advento do computador, pode-se usufruir de algoritmos que usam grandes chaves e separam a mensagem em pedaços, fazendo rotações e usando operações matemáticas como XOR, OR e ADD para embaralhar a mensagem original, tornando cada vez mais difícil sua descoberta.

A criptografia vem ganhando importância em vários setores, como, por exemplo, em aplicações militares e nos meios de transmissão de dados, em que a informação pode conter dados sigilosos, bancários, etc. Em resumo, onde possa haver pessoas mal-intencionadas a criptografia vem sendo usada abundantemente para garantir a qualidade da informação, bem como o sigilo.

Além dos meios de transmissão, existem casos em que é necessário proteger os dados que ficam armazenados em uma máquina local para evitar que informações possam ser descobertas por meio do roubo de dados, de forma ilegítima. É claro que nem todos sempre precisam de tal tecnologia, mas a proteção de dados é extremamente importante em algumas ocasiões. Essa necessidade fez surgirem os sistemas de arquivos criptografados, que são

sistemas de arquivos embaralhados por um certo algoritmo para que somente o detentor da senha possa compreendê-los.

Este trabalho tem como propósito fazer uma análise de dois métodos de criptografia, sendo um cifrando uma partição do disco e outro cifrando um arquivo, grande o suficiente, que armazenará outros arquivos dentro dele, como se fosse uma partição.

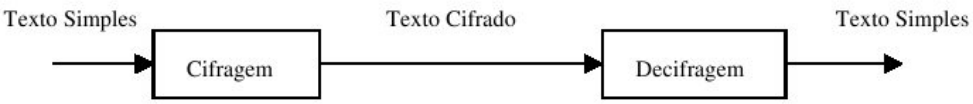
Para que essa apresentação seja feita, o texto está organizado como segue: o capítulo 2 conta um pouco da história da criptografia e discorre sobre os tipos de criptografia existentes; o capítulo 3 trata sobre os sistemas de arquivos e os sistemas de arquivos criptografados; o capítulo 4 mostra a implementação de dois tipos de criptografia de disco usando *loopback*; por fim, no capítulo 5 são apresentados os resultados obtidos.

2 O DESENVOLVIMENTO DA CRIPTOGRAFIA

A palavra criptografia vem do grego *kryptos*, 'escondido, oculto', e de *grafia*, no sentido de 'escrita', e pode ser definida como a arte ou a ciência de garantir a segurança de mensagens, de forma que apenas pessoas autorizadas a leiam (SCHNEIER, 1996). A criptografia pode ser usada em conjunto com outras técnicas para garantir confidencialidade, autenticidade, integridade e não-repúdio. Alguns termos específicos de criptografia são usados neste trabalho. Para auxiliar a leitura, apresentamos uma rápida explicação dos seus significados.

Uma mensagem é um texto simples. O processo de tornar seu conteúdo irreconhecível é chamado de *cifragem* ou *encriptação*. O processo contrário – ou seja, reverter e tornar o conteúdo novamente legível – é de chamado *decriptação* ou *decifragem* (Figura 1).

Figura 1 – Fluxo de uma mensagem usando criptografia



A mensagem, depois de encriptada, pode ser chamada de *criptografada* ou *cifrada*. Reconvertida ao conteúdo original, pode ser chamada de *decifrada* ou *decriptografada* (HINZ, 2005).

Um *transmissor* é aquele que transforma uma mensagem comum em uma mensagem criptografada e a envia para um *receptor*, que a recebe e a transforma novamente em uma mensagem comum.

A técnica de manter seguras as mensagens é chamada de *criptografia*. A técnica de tentar descobrir o conteúdo das mensagens cifradas é chamada de *criptoanálise* e seus praticantes, de *criptoanalistas* ou *atacantes*. O conjunto dessas duas técnicas é a chamada *criptologia*. Podem-se obter mais informações a esse respeito em Estany (2005), Braun (2005) e Hölzer (2005).

2.1 História da criptografia

Desde os seus primórdios, a humanidade em geral tem necessidade de guardar segredos. Algumas vezes esses segredos são importantes para a segurança de uma nação, outras vezes apenas para quem os guarda.

Muitos namorados, amantes e pessoas desejosas de sigilo vêm utilizando a criptografia através dos anos. No entanto, quem realmente investiu tempo e dinheiro a fundo em criptografia foram os militares, que dependiam da segurança para que suas mensagem chegassem intactas ao destinatário e sem que ninguém as tivesse lido no caminho. Vidas, guerras e muitas vitórias possivelmente foram decididas pelo uso da criptografia. Durante as guerras, não só os algoritmos foram usados exaustivamente pelos militares, como novos foram desenvolvidos. Alguns métodos antigos de cifragem de mensagens foram aperfeiçoados e aprimoraram-se os métodos de decifrá-las.

O primeiro relato de criptografia é conhecido como o algoritmo de César (WEBER, 1995), usado pelo imperador Júlio César na Roma Antiga. O algoritmo, simples, fazia substituições alfabéticas no texto da mensagem. As substituições aconteciam trocando as letras por outras, três posições à frente no alfabeto. Dessa forma, a letra A seria trocada pela letra D, a letra B seria substituída por E, C por F, D por G, e assim por diante.

Um exemplo clássico é:

Texto simples: **Vamos atacar o norte durante a noite**.
Texto cifrado: **Zdprv dxdfdu r qruxh gyudqxh d qrlxh**.

O algoritmo desenvolvido por César enganou seus inimigos por algum tempo, mas, depois de decifrado, passou a ser inútil. Uma pequena modificação nesse algoritmo produz um conceito de chave, utilizado até hoje, que consiste em substituir as letras por k posições, em vez de três posições à frente no alfabeto. Independentemente do algoritmo ser conhecido, a chave torna-se mais importante que o próprio algoritmo, pois sem ela não seria possível decifrar a mensagem.

O exemplo a seguir mostra um algoritmo de substituição que troca as letras 10 posições à frente (k=10).

Alfabeto simples: **abcdefghijklmnopqrstuvwxyz**.
Alfabeto cifrado: **klmnopqrstuvwxyzabcdefghij**.

Texto simples: **Vamos atacar o norte durante a noite**.
Texto cifrado: **Fkwyc kdkmkb y xybdo nebkxdo k xysyo**.

Vários algoritmos modernos usam um sistema muito similar de criptografia, no qual o algoritmo é conhecido mas a chave não, impossibilitando a decifragem da mensagem.

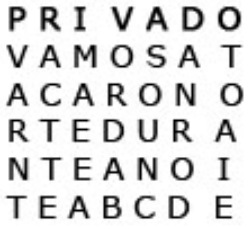
O tamanho da chave é um fator fundamental para o sigilo da transação. Se a chave tiver um tamanho 2 (dois dígitos), existe um universo de 100 combinações; para uma chave 6 (seis dígitos), existe um universo de 1 milhão de combinações. Dessa forma, quanto maior o número de dígitos da chave, maior será a segurança dessa chave.

Existem várias possibilidades no processo de criptografia, podendo ser citadas as cifras de substituição e as cifras de transposição. As cifras de

substituição são algoritmos que substituem letras por letras mas as mantêm na mesma posição, como o algoritmo de César. Em contrapartida, as cifras de transposição trocam a posição das letras sem trocar o conteúdo das letras, ou seja, somente embaralham a mensagem.

Um algoritmo de transposição conhecido é o de colunas (TANEMBAUM, 1997). Sua concepção determina que a chave seja uma palavra ou frase sem letras repetidas. Primeiramente escreve-se o texto preenchendo coluna por coluna, como se fosse uma matriz, e, finalmente, escreve-se o texto cifrado lendo a partir das colunas, e não a partir das linhas, seguindo a ordem alfabética de cada letra da palavra (Figura 2).

Figura 2 – Criptografia por transposição



Chave: **PRIVADO**
Texto simples: **Vamosatacaronorteduranteanoite**.
Texto cifrado: **Souncanrodmaeeatoaievarntactteordab**.

2.2 Tipos de criptografia

A criptografia atual é baseada em chaves. Uma chave é uma combinação de bits e, quanto maior for essa combinação, maior será a segurança obtida. Dependendo do tipo de chave utilizada, a criptografia classifica-se em *simétrica* e *assimétrica*.

A criptografia simétrica, como o nome sugere, baseia-se na simetria das chaves, ou seja, a mesma chave utilizada para criptografar será usada para descriptografar a mensagem (Figura 3). Essa chave, chamada privada, é previamente trocada entre o emissor e o receptor por um canal de comunicação seguro (VASQUES e SCHUBER, 2002).

Figura 3 – Criptografia simétrica

A desvantagem desse processo deve-se ao fato de que, como apenas uma chave é utilizada para cada par de pessoas, a segurança sobre ela deve ser rígida. Além disso, se o número de pessoas que queiram se comunicar de forma segura for muito grande, serão necessárias inúmeras chaves, o que dificultará ainda mais a sua gerência (VASQUES E SCHUBER, 2002).

Podem-se citar os seguintes algoritmos simétricos e os respectivos tamanhos das chaves geradas por cada um deles:

Data Encryption Standard (DES) – 56 bits.
Triple Data Encryption Standard (3DES) – 112 bits.
Advanced Encryption Standard (AES) – 128, 192 ou 256 bits.
Blowfish – até 448 bits.
Carlisle Adams and Stafford Tavares (CAST) – 128 ou 256 bits.
Twofish – 128, 192 ou 256 bits.
Serpent – 128, 192 ou 256 bits.

A criptografia assimétrica, por sua vez, envolve o uso de duas chaves distintas, uma privada e outra pública. Pode-se fazer uso de qualquer uma das chaves para cifrar a mensagem, mas somente a chave inversa deve ser utilizada para decifrá-la. Por exemplo, se um emissor utiliza a chave pública do receptor para cifrar a mensagem, o receptor deve utilizar a sua chave privada para decifrá-la (Figura 4). Dessa forma, garante-se autenticidade e confidencialidade (VASQUES E SCHUBER, 2002).

Contudo, se o emissor utilizar a chave privada para cifrar a mensagem, qualquer pessoa poderá decifrá-la, uma vez que a chave pública é de conhecimento de todos, o que garantirá apenas a autenticidade da mensagem.

Figura 4 – Criptografia assimétrica

Como exemplo de algoritmo assimétrico, pode-se citar o *Rivest Shamir Adleman* (RSA), composto de chaves de 512, 768, 1024 ou 2048 bits. Esse algoritmo é a base da maioria das aplicações de criptografia assimétrica em uso atualmente, pois seus mecanismos dificultam a obtenção da chave utilizada.

3 SISTEMAS DE ARQUIVOS

O sistema de arquivos é a parte mais visível do sistema operacional para o usuário. Ele deve apresentar uma interface simples, coerente e fácil de usar pois os usuários ao utilizar o computador, fazem constante uso dos sistemas de arquivos.

É importante salientar que os sistemas de arquivos implementam um recurso em *software* que não existe no *hardware*. O *hardware* oferece somente espaço em disco, na forma de setores e trilhas que podem ser acessados individualmente e em ordem aleatória.

O conceito de arquivo é muito mais útil que o simples espaço em disco, é uma abstração criada pelo sistema operacional. Neste caso, tem-se um recurso lógico a partir dos recursos físicos existentes no sistema operacional.

3.1 Conceitos básicos

Arquivos são recipientes que contêm dados. Em geral, cada arquivo contém um conjunto de dados que um usuário, por alguma razão, resolveu agrupar, segundo algum significado lógico para ele (SILVA, 2004). Um arquivo pode conter um programa executável, uma música, uma foto, etc.

Diretórios são conjuntos de referência a arquivos. Por exemplo, todos os arquivos de determinado usuário podem formar um diretório. Diretórios são úteis para organizar os arquivos do sistema, permitindo sua separação em grupos, o que facilita a localização e manuseio.

Arquivos são normalmente implementados a partir de discos magnéticos. Entretanto, existem situações nas quais é importante visualizar um único disco físico como se fossem vários. Um mesmo disco pode ser particionado em dois *discos lógicos*, ou *partições.* É possível manter todos os arquivos de sistema em uma partição e todos os arquivos de usuário em outra partição.

Cada partição é um disco autocontido. Dessa forma, caso ocorra um dano irreparável em alguns setores do disco, somente a partição que inclui aqueles setores será comprometida. As outras partições permanecerão intactas (Figura 5).

Figura 5 – Disco particionado

Como cada partição é, por definição, autocontida, nenhum arquivo poderá ser maior que uma partição inteira. Suponhamos que o computador tenha dois discos físicos de 5 gigabytes cada. Se cada disco for considerado uma partição, nenhum arquivo nesse sistema poderá ser maior que 5 gigabytes. Entretanto, caso seja criada uma partição composta dos dois discos físicos, pode-se ter um arquivo de até 10 gigabytes.

O conceito de partição ou disco lógico é, via de regra, implementado nos *device drivers*. Essa implementação é de tal forma que o sistema de arquivos não

tem condições de determinar se uma dada partição é composta de exatamente um disco físico, vários discos físicos ou parte de um único disco.

Na perspectiva do sistema de arquivos, cada partição corresponde a um conjunto de blocos físicos com tamanho fixo e numerados de 0 até N-1, onde N é o número de blocos físicos da partição. Cabe ao *device driver* mapear o endereço de bloco físico em uma posição específica dentro de um disco físico.

3.2 Controle de acesso

Em um sistema de multiusuários, é importante controlar o acesso aos arquivos. Por exemplo, um professor mantém um arquivo com as notas dos alunos e não gostaria que os alunos tivessem acesso a esse arquivo. Sistemas de arquivos modernos e sistemas operacionais multiusuários implementam mecanismos que permitem controlar quais usuários podem fazer o que em quais arquivos.

O controle de acesso tem início com a identificação dos usuários e é feito normalmente por meio de um código de usuário e uma senha. Antes de o usuário poder fazer qualquer coisa no sistema, seu código e sua senha devem ser validados, para que o sistema operacional saiba que o usuário naquele terminal é mesmo quem ele afirma ser. A partir do momento em que a identificação é feita, todos os processos disparados a partir do terminal em questão passam a ter os direitos de acesso associados a aquele usuário.

Entretanto, esse controle de acesso não é suficiente para garantir a privacidade e a segurança das informações. Qualquer pessoa que adquira o poder de superusuário pode facilmente sobrepor esse controle. Dessa maneira, os dados não estariam protegidos de administradores não-éticos e invasores de

sistemas. Os dados poderiam ser lidos com facilidade, por exemplo, ao se levar o disco para uma máquina preparada para isso.

Por esse motivo, a criptografia do sistema de arquivos torna-se uma alternativa não apenas interessante, mas necessária em casos em que os dados precisem estar protegidos de olhares indevidos. As seções a seguir apresentam algumas técnicas para atingir esse objetivo.

3.3 Criptografando o sistema de arquivos com o *kernel loopback*

Esse é o modo mais comum de usar encriptação em sistemas Linux. O *patch* do *loopback* é baseado no BSD *loopback encrypting*. O método permite que se monte um arquivo como um arquivo de sistema e que os dados armazenados dentro desse arquivo sejam menores que o próprio arquivo.

Para usar o *loopback encryption device* deve-se aplicar o *patch* no *kernel* e usar o comando *losetup*. Versões mais recentes do *kernel* vêm com esse *patch* aplicado.

A nova versão do *loopback encryption device* pode usar uma vasta gama de algoritmos criptográficos para embaralhar os dados, como DFC, MARS, RC6, Serpent, CAST 128, IDEA, Twofish, Blowfish, mas nem todos os algoritmos são suportados (TZEK, 2005).

3.4 Alternativas baseadas na arquitetura cliente/servidor

Algumas alternativas ao *kernel loopback* são apresentadas a seguir. Nenhuma delas é nativa do *kernel*, o que diminui a velocidade e aumenta um pouco a complexidade de sua implantação.

3.4.1 CFS – *Cryptographic File System*

O CFS, idealizado por Matt Blaze, foi o primeiro utilitário livre com a finalidade de encriptar o disco rígido local usando UNIX. Esse *hack* adicionou ao NFS uma nova função, que é o CFS. De fato, o usuário não precisa alterar o *kernel* para implementar essa funcionalidade, além do fato de o CFS ser mais portável que outras soluções UNIX.

Uma outra funcionalidade interessante é o usuário poder transmitir arquivos via NFS encriptados. O CFS suporta o algoritmo DES, o qual é inseguro por possuir uma chave curta; o 3DES pode ser considerado seguro mas muito lento; o MacGuffin pode ser quebrado (TZEK, 2005).

O maior problema do CFS é a lentidão. O resultado disso é que o CFS começa a usar o espaço do usuário e força o *daemon* a copiar os dados diversas vezes entre o espaço do *kernel* e o espaço do usuário (TZEK, 2005). Se o usuário desejar encriptar grandes quantidades de dados, o desempenho do sistema será penalizado se for usado o CFS.

Pode-se obter o código fonte do CFS em: http://koeln.ccc.de/archiv/drt/crypto/cfs-1.3.3.tar.gz

Para saber mais sobre o funcionamento do CFS, deve-se consultar "Cryptographic File System under Linux HOW-TO", ou "A Cryptographic File System for Unix", escrito por Matt Blaze.

3.4.2 TCFS – *Transparent Cryptographic Filesystem*

O TCFS foi desenvolvido pela Universidade de Salerno, Itália, para melhorar o CFS de Matt Blaze provendo uma integração profunda entre o *device* de encriptação e o sistema de arquivos, obtendo-se como resultado uma

completa transparência para o usuário. No entanto, os desenvolvedores parecem ter visualizado melhor que Matt Blaze ao substituir o NFS.

Uma nova funcionalidade do TCFS é permitir um compartilhamento seguro de arquivos entre membros de um mesmo grupo. Uma das grandes mudanças é o TCFS precisar de um *patch* aplicado no *kernel*.

Mais informações sobre o TCFS podem ser obtidas no TCFS-FAQ. Pode-se obter o TCFS a partir da *home page* do projeto.

3.4.3 ppdd – *Practical Privacy Disc Driver*

O ppdd é um outro meio de encriptar discos. O autor Allan Lathan assim o define:

> PPDD ajuda você a encriptar o sistema de arquivos em Linux. Ele usa técnicas de alta qualidade para grandes volumes. É fácil de instalar e usar. Proteção é para toda a máquina. Tão logo o root ative o driver o sistema de arquivos torna-se disponível usando a permissão de um usuário. Isso é ideal para máquinas que têm apenas um usuário, ou apenas um componente a mais de segurança para máquinas multiusuários. A performace é inferior a um sistema de arquivos não encriptado, mas um pentium 100MHz com 32Mb de memória principal usando um disco IDE reduz a vazão de dados em 50%. Isso deve ser aceitável considerando a criptografia por software.

Uma das coisas interessantes do ppdd é o usuário poder decriptografar *devices* sem o suporte do *kernel*. O autor pensou exaustivamente em meios de fazer *backup* do sistema de arquivos encriptado e apresentou diversas soluções para resolver esse problema. Ele sugeriu até a possibilidade de ter a partição raiz encriptada.

Um dos inconvenientes do ppdd é usar o Linux crypto-API, ou seja, não é nativo do *kernel* do sistema, o que dificulta um pouco sua implementação, pois é necessário recompilar o *kernel*. Por não ser nativo do *kernel* o ppdd não foi

utilizado neste trabalho, cujo objetivo é explorar um meio de partição fácil de implementar, eficiente e que ofereça uma boa criptografia.

Mais informações a respeito do ppdd podem ser obtidas nas *man pages* .

3.5 Comentários finais

Este trabalho adota o método de encriptação de disco por *loopback*, pois, além de fazer parte nativamente do *kernel*, é um algoritmo rápido, eficiente e relativamente simples.

A implementação desse algoritmo não leva mais de 10 minutos se for usada a opção de criptografar um arquivo grande o suficiente, não havendo a necessidade de reparticionar o disco rígido.

O método prático é descrito no capítulo 4, que contém alguns exemplos da solução.

4 INSTALAÇÃO/CONFIGURAÇÃO DE UM ARQUIVO COMO DISPOSITIVO DE BLOCO

Nos testes com criptografia de disco e criptografia de bloco foi usado um computador HP, com um processador Intel Pentium IV de 2600 megahertz, 1024 MB de memória RAM e dois discos rígidos SCSI, sendo um de 133 gigabytes e outro de 32 gigabytes.

O sistema operacional foi o Fedora Core 4, não tendo sido abordada a instalação do sistema operacional neste trabalho. Essa distribuição foi usada porque possui os requisitos necessários nativamente compilados no *kernel*.

4.1 *Kernels* e módulos necessários

A versão do *kernel* utilizada foi a 2.6.11-1. São necessárias as seguintes funções para a utilização da solução criptográfica, como módulos ou integradas ao *kernel*:

- *loop*: permite o uso de sistemas de arquivos em *loopback.*
- *blowfish*: algoritmo de criptografia usado. Poderia ser usado qualquer outro, como AES, DES ou mesmo *twofish*.
- *cryptoloop*: permite o uso de sistemas de arquivo em *loopback* com encriptação.

É necessário também que alguns módulos do *kernel*, que são os algoritmos criptográficos, estejam compilados. Como os módulos somente lançam mão da memória quando são utilizados, podem-se compilar todos os algoritmos disponíveis para que no futuro existam maiores opções de criptografia.

As seguintes configurações são necessárias no *kernel*, caso seja necessário recompilá-lo:

```
#
# BLOCK DEVICES
#

CONFIG_BLK_DEV_LOOP=m
CONFIG_BLK_DEV_CRYPTOLOOP=m

#
# CRYPTOGRAPHIC OPTIONS
#

CONFIG_CRYPTO=y
CONFIG_CRYPTO_HMAC=y
CONFIG_CRYPTO_NULL=m
CONFIG_CRYPTO_MD4=m
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_SHA1=m
CONFIG_CRYPTO_SHA256=m
CONFIG_CRYPTO_SHA512=m
CONFIG_CRYPTO_DES=m
CONFIG_CRYPTO_BLOWFISH=m
CONFIG_CRYPTO_TWOFISH=m
CONFIG_CRYPTO_SERPENT=m
CONFIG_CRYPTO_AES=m
CONFIG_CRYPTO_CAST5=m
CONFIG_CRYPTO_CAST6=m
CONFIG_CRYPTO_ARC4=m
CONFIG_CRYPTO_DEFLATE=m
CONFIG_CRYPTO_MICHAEL_MIC=m
CONFIG_CRYPTO_CRC32C=m
# CONFIG_CRYPTO_TEST não deve ser setado
```

O Fedora Core 4 suporta criptografia e não precisa ser recompilado. Caso seja necessário recompilar o *kernel* por algum motivo, como desempenho ou item específico, basta adicionar os módulos acima para habilitar a parte criptográfica.

Se o usuário desejar saber se o *kernel* de outra distribuição aceita encriptação nativa, deve ler o manual da distribuição. Não havendo manual

pode-se tentar carregar os módulos com *modprobe* módulo ou procurar os módulos nos diretórios /lib/modules/2.4.x para *kernels* da versão 2.4 ou /lib/modules/2.6.x para *kernels* da versão 2.6.

4.1.1 Carregando os módulos necessários para a memória

Depois de compilados os módulos do *kernel*, é possível carregar os módulos na memória para que o sistema operacional possa interagir com eles da seguinte forma:

```
# modprobe cryptoloop
# modprobe blowfish
# modprobe loop
```

Normalmente o modulo *loop* é carregado no *boot* da maior parte das distribuições. Para carregar os outros módulos automaticamente, pode-se adicioná-los ao /etc/modules.

4.1.2 Criando e montando um dispositivo de bloco criptografado

Dentre as duas formas de implementar criptografia de disco demonstradas neste trabalho, encriptar um dispositivo de bloco é o método mais fácil. Não é necessário reparticionamento, uma vez que todo o dispositivo fica armazenado em um arquivo normal do sistema.

O sistema é muito seguro e só poderá ser montado por quem souber a senha usada no momento da encriptação. Usando senhas longas, é praticamente impossível descobrir a senha correta por força bruta.

Deve-se criar um arquivo com o tamanho desejado para o espaço de arquivos criptografados. É um arquivo comum, mas internamente ele será

cifrado e possuirá uma organização de sistemas de arquivos. Pode-se criar o arquivo com o comando dd. Além disso, para melhorar a segurança, pode-se criar o arquivo a partir de dados randômicos com o seguinte comando:

 # dd if=/dev/random of=/var/arquivocifrado bs=1M count=1000

Com esse comando cria-se um aquivo com conteúdo randômico de 1 gigabyte dentro do diretório /var chamado *arquivocifrado*. Esse arquivo pode ter qualquer nome, com qualquer extensão: o importante é o conteúdo.

Em seguida, deve-se associar o arquivo a um dispositivo de bloco e criptografá-lo usando um dos algoritmos conhecidos pelo *kernel* para criar a partição criptografada.

O comando responsável por essa associação é o *losetup*, por exemplo:

 # losetup -e blowfish /dev/loop0 /var/arquivocifrado
Password:

A opção -e informa ao *losetup* que ele deve criptografar o arquivo /var/arquivocifrado usando o algoritmo *blowfish* e associá-lo ao /dev/loop0.

O *password* é pedido uma vez. Essa senha é necessária para acessar a partição, portanto deve-se ter muito cuidado com ela, pois é impossível trocá-la sem formatar novamente o sistema de arquivos.

Agora, pode-se formatar o dispositivo e criar o sistema de arquivos propriamente dito, por exemplo:

 # mkfs.ext2 /dev/loop0

Pode-se formatar o dispositivo com o sistema de arquivos que mais convier, pois o dispositivo se comporta como se fosse uma partição separada do disco. Após formatado, pode-se montar o arquivo como uma partição normal, por exemplo:

 # mount -t ext2 /dev/loop0 /mnt/segredo

Com isso, agora pode-se navegar no diretório e gravar arquivos normalmente como qualquer parte do sistema. Quando o usuário terminar as atividades e desejar desligar a máquina, pode desmontar o dispositivo utilizando os comandos:

 # umount /mnt/segredo
 # losetup -d /dev/loop0

A opcao -d do *losetup* faz com que ocorra a desassociação do /dev/loop0 do /var/arquivocifrado.

Para associar novamente o /dev/loop0 ao /var/arquivocifrado deve-se fornecer a senha digitada anteriormente no momento da criação da partição. O conteúdo do arquivo não será lido até que seja informada a senha correta.

4.2 Criando uma partição de arquivos criptografada

A encriptação de uma partição é mais simples do que encriptar o disco todo, pois ajuda a manter os dados sigilosos de forma muito segura. No entanto, corre-se o risco de dados serem copiados em outras áreas não criptografadas do disco.

Adicionalmente, além da partição de dados, é interessante criptografar também a partição *swap*, para evitar que qualquer dado importante armazenado possa ser recuperado de alguma forma (FERREIRA, 2005). A criptografia da *swap* não é discutida neste trabalho.

De forma muito parecida com a criação de um bloco criptografado, é possível criar uma partição criptografada, que pode ser um outro disco rígido, uma partição no mesmo disco ou um *pendrive*. Além disso, deve-se associar um

dispositivo à partição para poder criptografá-la, procedendo da mesma forma como foi feito para o dispositivo de bloco, usando o *losetup*:

 # losetup -e blowfish /dev/loop0 /dev/hda2
Password:

É necessário informar uma senha para acessar o compartilhamento, lembrando que a troca da senha é impossível sem reformatar a partição e que sem a senha não é possível acessá-la. Após vinculado, o *device loopback* poderá ser manipulado como um *block device* comum (SLACKWARE ZINE). Em seqüência, deve-se então formatar a partição:

 # mkfs.ext2 /dev/loop0

Pode-se formatar o dispositivo com o formato que mais convier, como ext2, ext3, reiserfs, xfs, etc. Após formatado, deve-se montar o dispositivo normalmente, de forma similar ao dispositivo por bloco:

 # mount -t ext2 /dev/loop0 /mnt/segredo

Após esse procedimento, é possível gravar arquivos e trabalhar normalmente com a partição, como outra qualquer.

Quando não se desejar mais usar a partição ela deve ser desmontada por meio do seguinte procedimento:

 # umount /mnt/segredo
 # losetup -d /dev/loop0

A opção -d é usada para desconectar a partição /dev/hda2 do dispositivo /dev/loop0.

4.3 Automatizando a tarefa de montar e desmontar a partição

No Anexo 1, é disponibilizado um *script* que automatiza a tarefa de montar a partição critografada, pedindo senha ao iniciar o sistema operacional e desmontando-a ao desligar o computador caso seja chamado no *boot* da máquina. Se adicionado ao *boot*, ele pode facilitar muito a vida do usuário, pedindo a senha apenas uma vez para montar e desmontando-a ao desligar a máquina. Pode também ser usado como um serviço e ser iniciado e parado a qualquer momento.

O *script* foi feito por Leandro Padilha Ferreira (2005) em seu estudo "Implementação de um sistema de arquivos criptografado transparente ao usuário" e alterado para a realidade deste trabalho.

O programa agrupa todos os passos necessários para a montagem da partição, sendo a senha necessária somente no momento do *boot*, caso seja adicionado ao sistema de *boot*.

Essa automação traz a vantagem da segurança. Caso o computador ou o *hard disk* seja roubado, se o ladrão não souber a senha a partição não montará. No entanto, essa segurança também tem desvantagens, porque, caso a máquina seja rebotada remotamente, a inicialização irá parar, pedindo a senha para partição, como normalmente acontece (caso adicionado ao sistema de *boot*), e até receber uma senha, correta ou não, a inicialização ficará congelada. Se for inserida uma senha errada, a inicialização continua normalmente, mas a partição criptografada não é montada.

No Anexo 2 encontra-se o *script* de "Automação de uso de sistemas de arquivos criptografados", que facilita a montagem/desmontagem do arquivo criptografado como um diretório. A única diferença, se comparado ao *script* do Anexo 1, é o parâmetro arquivo: deve ser atribuído o nome do arquivo criptografado com sua devida localização.

O *script* é também de autoria de Leandro Padilha Ferreira (2005) em seu trabalho “Implementação de um sistema de arquivos criptografado transparente ao usuário”, tendo sido alterado para a realidade deste trabalho.

4.4 Testes de performance

Para o teste de performance, foi criada uma partição cifrada de 10 gigabytes e um arquivo de 9 gigabytes, juntamente com um *script* para automatizar a tarefa, que está descrito no Anexo 3. O arquivo de 9 gigabytes localiza-se em outro disco rígido da mesma máquina. Ambos os discos rígidos são SCSI e usam formatação ext3, inclusive o sistema criptografado.

Cada teste foi executado pelo menos três vezes, e o valor apresentado foi a média dos valores. Depois de cada teste a máquina foi reiniciada, para evitar que os dados pudessem usar cache para agilizar a cópia, mostrando valores errados.

O procedimento do *script* é:

- Guarda a hora atual inicial;
- Faz a cópia do arquivo de 9 gigabytes para um arquivo cifrado;
- Guarda a hora atual final;
- Diminui a hora inicial da hora final.

Com isso, pode-se obter a diferença de tempo entre copiar um arquivo grande para um arquivo cifrado e para uma partição normal. De posse dessa informação, pode-se calcular a performance comparativa entre as duas partições.

Os resultados obtidos podem ser vistos na Tabela 1.

Tabela 1 – Resultados obtidos com testes de grandes arquivos

	Arquivo criptografado	**Partição Ext3**
Copiar arquivo de 9 gigas	9 minutos e 57 segundos	5 minutos e 8 segundos
Mover arquivo de 9 gigas	10 minutos e 15 segundos	5 minutos e 10 segundos

No segundo teste foi usado o "/usr" contendo 6836 arquivos em 1.6 gygabytes de dados. Foi usado o mesmo script do Anexo 3 de forma adaptada.

A Tabela 2 mostra os resultados do segundo teste.

Tabela 2 – Resultados obtidos com testes de pequenos arquivos

	Arquivo criptografado	**Partição Ext3**
Copiando o diretório /usr	5 minutos e 45 segundos	4 minutos e 38 segundos
Movendo o diretório /usr	6 minutos e 38 segundos	5 minutos e 15 segundos

O terceiro teste consistiu em copiar o arquivo de 9 gigabytes para a partição encriptada e rebootar o servidor a fim de simular uma queda de energia.

O resultado obtido foi que a partição não foi corrompida, tendo montado normalmente. No entanto, o arquivo que estava sendo copiado ficou pela metade, mas mesmo assim não afetou os outros arquivos da partição encriptada.

Para certificar que a partição estava correta, foi aplicado o fsck, que trabalhou de modo transparente, igual a uma partição normal.

5 CONCLUSÃO

Cada vez mais se torna necessário proteger a informação de pessoas mal-intencionadas. Existem vários métodos de proteção para todo o tipo de ataques, mas quando os ataques chegam ao nível físico, como roubar a máquina, fica difícil negar o acesso aos dados.

Para conter o acesso físico, foi criada a criptografia de disco, na qual os arquivos ficam em uma partição cifrada, que só pode ser acessada mediante o uso da senha correta. Essa partição contém dados embaralhados por um algoritmo criptográfico conhecido e suportado pelo *kernel*.

Com as configurações corretas, não é difícil produzir uma partição encriptada e *scripts* para monitorar a montagem e a desmontagem com controle de senha, tornando as partições criptografadas simples e fáceis de usar.

Uma vez que os dados estão guardados dentro da partição cifrada, a única forma de visualizar o conteúdo é com a máquina ligada e a senha correta. Sem essa senha, não é possível descobrir o conteúdo. Dessa forma, uma grande segurança é adicionada em caso de roubos de computadores e de discos rígidos para obter informação alheia.

Uma desvantagem desse tipo de solução é que, se a partição está montada em um computador móvel, os dados ficam expostos. Mas pode-se fazer um *script* para ser rodado em momentos de pânico, quando ele desmonta a partição, protegendo os dados.

Outra desvantagem dessa solução é os dados trafegarem pela partição *swap* ou mesmo pela /tmp ou /var. O ideal é criptografar tudo, mas mesmo assim ainda existem *pendrivers*, CD-ROM e disquetes onde as informações trafegam de forma não criptografada, podendo gerar uma brecha na segurança.

No que se refere à performance, a perda de velocidade não foi desprezível, mas o ganho na área de segurança foi satisfatório. Cabe ao usuário decidir se o custo computacional vale a perda de performance quando for implementar uma solução.

Se o item primordial for rapidez, essa solução não é indicada; se for segurança, atende aos requisitos necessários.

REFERÊNCIAS BIBLIOGRÁFICAS

CRYPTOGRAPHIC File System for Unix. Disponível em: <http://www.crypto.com/papers/cfs.pdf>. Acesso em: 12 nov. 2005.

CRYPTOGRAPHIC File System under Linux HOW-TO. Disponível em: <http://koeln.ccc.de/archiv/drt/crypto/cfs-linux-HOWTO.txt>. Acesso em: 10 nov. 2005.

BRAUN, David. **Disk Encryption HOWTO**. Disponível em: <http://tldp.org/HOWTO/Disk-Encryption-HOWTO/>. Acesso em: 1 nov. 2005.

ESTANY, C. P. **Cifrando un sistema de ficheros**. Disponível em: <http://bulma.net/impresion.phtml?nIdNoticia=1970>. Acesso em: 1 nov. 2005.

FEDORA Core. Disponível em: <http://fedora.redhat.com>. Acesso em: 10 ago. 2005.

FERREIRA, L. P. **Implementação de um sistema de arquivos criptografado transparente ao usuário**. Disponível em: <http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=1513>. Acesso em: 3 nov. 2005.

HINZ, M. A. M. **Um estudo descritivo de novos algoritmos de criptografia**. Disponível em: <http://www.ufpel.edu.br/prg/sisbi/bibct/acervo/info/2000/Mono-MarcoAntonio.pdf>. Acesso em: 7 nov. 2005.

HÖLZER, R. **Cryptoloop HOWTO**. Disponível em: <http://www.tldp.org/HOWTO/Cryptoloop-HOWTO/index.html>. Acesso em: 31 out. 2005.

PPDD man pages. Disponível em: <http://koeln.ccc.de/archiv/drt/crypto/ppdd.man.html>. Acesso em: 20 nov. 2005.

SCHNEIER, B. **Applied cryptography**. 2nd ed. EUA: John Wiley & Sons, 1996. 758 p.

SILVA, R. M. de A. **Sistemas operacionais**. Lavras: UFLA, 2004.

SLACKWARE ZINE: **File systems criptografados**. Disponível em: <http://www.slackwarezine.com.br/download/slackzine6.pdf>. Acesso em: 3 nov. 2005.

TANEMBAUM, A. **Redes de computadores.** 1997.

TCFS home page. Disponível em: <http://www.tcfs.it/>. Acesso em: 7 nov. 2005.

TCFS-FAQ. Disponível em: <http://www.tcfs.it/docs/tcfslinux-faq.3.html>. Acesso em: 1 dez. 2005.

TZEK, D. R. **Encrypting your disks with Linux**. Disponível em: <http://koeln.ccc.de/archiv/drt/crypto/linux-disk.html>. Acesso em: 3 nov. 2005.

VASQUES, A. T., SCHUBER, R. P. **Implementação de uma VPN em Linux utilizando o protocolo IPSEC**. Disponível em: <http://www.abusar.org/manuais/VPN-alan-rafael.pdf>. Acesso em: 28 nov. 2005.

WEBER, R. F. **Criptografia contemporânea**. Congresso Brasileiro de Computação. Canela, 1995.

ANEXOS

Anexo 1 – Script de automatização para montar e desmontar a partição

```
#!/bin/sh
#
# Monta partição criptografada hda6
#
# Criado por: Leandro Padilha Ferreira
# Adaptado por: Eduardo Paulo Gasparetto em 3/11/2005
#


# particao criptografada
PARTICAO=/dev/hda2

# dispositivo usado, /dev/loopX
DEVLOOP=/dev/loop0

# Ponto de montagem, lugar onde ficará disponível a particao
PONTOMONTAGEM=/mnt/segredo

# algoritmo de encriptacao usado
ALGORITMO=blowfish

# mensagens do programa
MSGPREPAR="Preparando-se para montar $PONTOMONTAGEM"
MSGNOROOTUSER="Você deve ser o super-usuário para executar essa ação."
MSGOK="Montado ok!"
MSGERROR="Não foi possivel montar $PONTOMONTAGEM"

# testa para ver se o modulo foi carregado, se nao foi entao carrega
lsmod | grep $ALGORITMO 1>/dev/null 2>/dev/null
if [ $? != 0 ];then
   modprobe $ALGORITMO
   sleep 1
fi

# testa para ver se o cryptoloop esta carregado, se nao esta entao carrega
lsmod | grep cryptoloop 1>/dev/null 2>/dev/null
if [ $? != 0 ];then
   modprobe cryptoloop
   sleep 1
```

```
fi

# se nao estiver sendo executado pelo root, então sai do script
if [ "$UID" != "0" ]
then
  echo $MSGNOROOTUSER
  exit 1
fi

# Funcao para montar dispositivos ou particoes
montar()
{
   echo $MSGPREPAR
   losetup -e $ALGORITMO $DEVLOOP $PARTICAO
   if [ $? == 0 ];then
    mount -t auto $DEVLOOP $PONTOMONTAGEM && df -Th | grep
$DEVLOOP && echo \\
 $MSGOK && exit 0
   fi
}

# Funcao para desmontar dispositivos ou particoes
desmontar()
{
   umount $PONTOMONTAGEM
   losetup -d $DEVLOOP && df -Th |grep $DEVLOOP || echo "Desmontado
ok!" && exit 0
   exit 1
}


case "$1" in
  start)
     montar
     ;;
  stop)
     desmontar
     ;;
  *)
     echo "Uso: /etc/init.d/hda2 {start|stop}"
     exit 1
esac
```

```
exit 0
```

Anexo 2 – Automatizando a tarefa de montar e desmontar o dispositivo de bloco

```
#!/bin/sh
#
# Monta partição criptografada hda6
#
# Criado por: Leandro Padilha Ferreira
# Adaptado por: Eduardo Paulo Gasparetto em 3/11/2005
#

ARQUIVO=/var/arquivocifrado

montar()
{
   losetup -e $ALGORITMO $DEVLOOP $ARQUIVO
   if [ $? == 0 ];then
      mount -t ext2 $DEVLOOP $PONTOMONTAGEM && df -Th | grep
$DEVLOOP && echo \\
 $MSGOK && exit 0
   fi
   exit 1
}

desmontar()
{
   umount $PONTOMONTAGEM
   losetup -d $DEVLOOP && df -Th |grep $DEVLOOP || echo "Desmontado
ok!" && exit 0
   exit 1
}
```

Anexo 3 – Script para teste de performance

```
#!/bin/bash
# 01/12/2005
# Desenvolvido por: Eduardo Paulo Gasparetto
#
#

echo "iniciando a copia do arquivo para a partição cifrada..."
horainicial=`date +%s`
cp /dados/arquivao /cifrada
horafinal=`date +%s`
tempocopia=`expr $horafinal - $horainicial`
echo "Copia para a particao cifrada foi efetuada em de $tempocopia segundos"
################
echo "iniciando a copia do arquivo para a partição normal..."
horainicial=`date +%s`
cp /dados/arquivao /normal
horafinal=`date +%s`
tempocopia=`expr $horafinal - $horainicial`
echo "Copia para a particao normal foi efetuada em de $tempocopia segundos"
#####
echo “Feito.”
```