LUCIANA MOREIRA SÁ DE SOUZA

MOBSEC - ADAPTAÇÃO DE SERVIÇOS DE ARMAZENAMENTO SEGURO PARA DISPOSITIVOS MÓVEIS

FLORIANÓPOLIS
2005

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

MobSEC - Adaptação de Serviços de Armazenamento Seguro para Dispositivos Móveis

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

LUCIANA MOREIRA SÁ DE SOUZA

Florianópolis, Fevereiro de 2005.

MOBSEC - ADAPTAÇÃO DE SERVIÇOS DE ARMAZENAMENTO SEGURO PARA DISPOSITIVOS MÓVEIS

Luciana Moreira Sá de Souza

'Esta Dissertação foi julgada adequada para a obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

_____
Prof. Joni da Silva Fraga, Dr.
Orientador

_____
Denizar Cruz Martins, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____
Prof. Joni da Silva Fraga, Dr.
Presidente

_____
Prof. Lau Cheuck Lung, Dr.
Co-orientador

_____
Prof. Ricardo Felipe Custódio, Dr.

_____
Prof. Rômulo Silva de Oliveira, Dr.

_____
Prof. Vitório Bruno Mazzola, Dr.

Ao Ricardo, por todo seu
amor, carinho e compreensão...

Agradecimentos

A Deus por ter me dado força e saúde para a conclusão deste trabalho.

À Universidade Federal de Santa Catarina, e ao PGEEL por ter me proporcionado a oportunidade de realizar este trabalho.

Ao Cnpq por ter me concedido uma bolsa de estudos no primeiro ano do mestrado.

Ao meu orientador Joni da Silva Fraga e co-orientador Lau Cheuck Lung, pela paciência, esforço e sugestões, apesar das dificuldades e distância. Agradeço ao meu co-orientador Jochen Haller (Alemanha), pela orientação e amizade, indispensáveis para a conclusão deste trabalho.

Agradeço a toda minha família e amigos, por terem compreendido minha ausência e estresse, em especial a minha amiga Érika pelos sábios conselhos.

Aos meus cunhados Rafael e Régis e a turma do RPG, em especial ao Leivas e Gustavo, pelas noitadas que ajudaram a aliviar o estresse. Ao meu sogrão José Carlos pelo respeito e pelos vinhozinhos sempre na hora e temperatura ideais. À minha sogrinha Marilia por ser compreensiva e por me acolher como filha.

À minha avó por ser tão carinhosa e colocar Deus em minha vida, rezando sempre por mim. À minha irmã que apesar de estar muito longe sempre me incentivou para prosseguir, estando muitas vezes mais preocupada comigo que eu mesma.

Aos meus pais Otávio e Eliane, pelo amor, compreensão e auxílio dados durante toda minha vida, em especial na reta final deste trabalho e principalmente por terem me proporcionado um ensino de qualidade, não medindo esforços e sacrifícios.

Ao Ricardo, meu esposo, companheiro e amigo que está sempre ao meu lado nas horas mais difíceis me auxiliando a enxergar adiante não me deixando desistir de meus objetivos.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

MOBSEC - ADAPTANDO SERVIÇOS DE ARMAZENAMENTO SEGURO PARA DISPOSITIVOS MÓVEIS

Luciana Moreira Sá de Souza

Fevereiro 2005

Orientador: Joni da Silva Fraga, Dr.
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Segurança computacional, armazenamento seguro, dispositivos móveis
Número de Páginas: 89

Os avanços tecnológicos em sistemas de comunicação *wireless* e dispositivos móveis estão criando um novo campo de aplicações onde uma maior quantidade de dados pode ser armazenada. Ainda que os avanços tecnológicos tenham eliminado ou reduzido os limites, outros aspectos devem, agora, ser levados em consideração. Estas aplicações para dispositivos móveis armazenam dados sensíveis da empresa e de clientes e devem, portanto, ser protegidos contra ataques e ameaças como roubo ou perda, acessos não autorizados, uso e distribuição dos dados. Para combater estes riscos é necessário armazenar os dados de forma segura e transparente para o usuário final. Finalmente, num cenário multi-usuário, onde diversos funcionários da empresa compartilham o mesmo aparelho, a confidencialidade dos dados deve ser mantida para dados do usuário e dados de grupos de usuários. As soluções disponíveis para proteger informação armazenada, não são adequadas para dispositivos móveis, pois ou usam algoritmos que consomem muito recurso de CPU, ou não são portáveis para estes aparelhos. O *MobSec* é uma arquitetura projetada para fornecer a proteção necessária para os dados armazenados em dispositivos móveis. Tendo sido especialmente desenhado para estes aparelhos, esta arquitetura leva em consideração as limitações e diversidade dos mesmos além dos riscos a que estes estão expostos.

Abstract of Dissertation presented to UFSC as one of the requirements for the degree of Master in Electrical Engineering.

MOBSEC-ADAPTING SECURE STORAGE SERVICES FOR MOBILE DEVICES

Luciana Moreira Sá de Souza

February 2005

Advisor: Joni da Silva Fraga, Dr.
Area of Concentration: Control, Automation and Industrial Computing
Keywords: Computer Security, secure storage, mobile devices
Number of Pages: 89

The technological advancements on wireless communication systems and mobile devices are creating a new field of business applications capable of storing of greater amount of data. Still, even if the technological advancements eliminated or pushed limits, other concerns must now be addressed. In these mobile business applications, sensitive data from the company and from the costumers is stored on a mobile device. They must, therefore, be protected against threats and attacks such as theft or lost of the device, unauthorized access, use and distribution of data. To address these risks, sensitive data must be stored in a secure and transparent way for the end user. Additionally, in a multi-user scenario where several employees share the same device, confidentiality of data must be granted for user files as for group files. In all cases, the company server must be able of recovering the data to process it. Available solutions for protecting stored data are not adequate for mobile devices since they use algorithms which are highly CPU consuming, are not portable. MobSec is an architecture designed to provide the necessary protection for data stored on mobile devices in case of theft of the device, false data injection and data tampering. Since this system was specially designed for mobile devices, concerns such as its limitations, diversity and the risks they are exposed to were taken into account.

Sumário

Lista de Ilustrações

Lista de Tabelas

Lista de Siglas

ACL	Access Control Lists
CFS	Cryptographic File System
EFS	Encrypting File System
PDA	Personal Digital Assistant
SGCC	Sistema de Gerenciamento de Chaves no Cliente
SGCS	Sistema de Gerenciamento de Chaves no Servidor
SRAF	Secure Random Access File
TCFS	Transparent Cryptographic File System
B2E	Business to Employee
ID	Identificação

Capítulo 1: Introdução

Com o aumento do uso de aplicativos, os dispositivos móveis estão evoluindo do seu tradicional papel como aparelhos apenas para conversa com funcionalidades limitadas, para dispositivos computacionais capazes de realizar uma grande variedade de funções. O desenvolvimento de processadores mais rápidos e aumento da capacidade de memória tem tornado telefones celulares e PDAs cada vez mais importantes em aplicações de negócios. Hoje os dispositivos móveis já são capazes de armazenar uma grande quantidade de dados, sejam eles pessoais, da empresa ou de terceiros.

À medida que as aplicações se tornam mais complexas, aumenta a necessidade de prover soluções de segurança para estes aparelhos. Dispositivos móveis estão muito sujeitos a roubo e perda, muitas vezes mais do que os *desktops*. Os dados armazenados nestes dispositivos ficam vulneráveis ao acesso, uso ou distribuição não autorizados, tornando-se indispensável protegê-los.

Diversos sistemas que visam prover a segurança dos dados armazenados têm sido desenvolvidos, sendo o CFS [4] o pioneiro entre eles. Estes sistemas, denominados sistemas de armazenamento seguro de dados, quando são aplicados aos dispositivos móveis, a conjugação usabilidade-desempenho-segurança não pode ser articulada, por conta de restrições apresentadas em suas concepções, conforme constatado nas arquiteturas TCFS, EFS e NCryptFS entre outras, analisadas em [30]. Por exemplo, implementar formas típicas de segurança em dispositivos móveis, usando infra-estrutura de chave pública própria de algumas arquiteturas não é adequado, pois são necessários algoritmos criptográficos que consomem muito recurso de CPU.

Faz-se, portanto, necessária a criação de uma arquitetura para armazenamento seguro destes dados, buscando atender às necessidades dos usuários de dispositivos móveis, ao mesmo tempo em que considera as limitações ainda existentes nestes dispositivos.

1.1. Objetivo e Justificativa

Este trabalho tem como objetivo principal propor a arquitetura MobSec para armazenamento seguro de dados em dispositivos móveis, garantindo a conjugação dos fatores usabilidade, desempenho, integridade e confidencialidade. Especificamente este trabalho pretende:

- Apresentar um estudo sobre segurança de dados em dispositivos móveis;
- Estudar técnicas de armazenamento seguro;
- Introduzir um modelo de suporte de armazenamento seguro que atenda a necessidade de manter a inviolabilidade de informações críticas;
- Desenvolver política que verifique a viabilidade das propostas referentes ao modelo projetado;
- Prover confidencialidade e integridade dos dados nos dispositivos móveis.

Os estudos realizados nesta pesquisa estão concentrados em tecnologias emergentes em que a literatura ainda não está consolidada no que se refere a padrões. Espera-se que o trabalho de certa forma contribua para a construção de dispositivos móveis mais seguros.

1.2. Estrutura do Trabalho

O trabalho está organizado em sete capítulos, abordando conteúdos de fundamentação teórica, modelo proposto e bibliografia utilizada na pesquisa.

Neste capítulo foram apresentados a contextualização do problema, os objetivos e as justificativas da pesquisa.

Os capítulos 2 e 3 consistem na revisão da literatura. O capítulo 2 trata da segurança em dispositivos móveis, apresentando os conceitos fundamentais de segurança e as principais características destes dispositivos. O capítulo 3 apresenta os principais sistemas de armazenamento seguro de dados realizando uma comparação entre eles.

Os capítulos 4 e 5 referem-se ao projeto MobSec. No capítulo 4 são apresentados a arquitetura proposta e seus componentes, funções e comportamento; no capítulo 5 são registrados aspectos da implementação e uma análise dos resultados obtidos.

O capítulo 6 apresenta as conclusões da pesquisa e o capítulo 7 encerra o trabalho com a relação da bibliografia utilizada.

1.3. Limitação da Pesquisa

A arquitetura do sistema MobSec proposto nesta pesquisa é dividida em dois módulos, Cliente e Servidor, com foco no primeiro. A arquitetura do Módulo Servidor é especificada somente nos aspectos necessários para permitir a criação e recuperação de chaves de usuários. Foge do escopo desta pesquisa, a definição dos demais componentes do Módulo Servidor.

Capítulo 2: Segurança em Sistemas Computacionais

Apesar do uso original dos computadores ter sido relacionado com a pesquisa avançada e a estratégia militar, a segurança de tais dispositivos era primeiramente física: diversos pontos de verificação e barreiras físicas tinham que ser transpostas antes de se conseguir acesso a um terminal.

Hoje, dispositivos computacionais com capacidades semelhantes, senão superiores, aos computadores usados originalmente para fins estratégicos, podem ser adquiridos a preços acessíveis. Dada esta disponibilidade, cada vez mais aplicações complexas estão sendo trazidas para o ambiente móvel: porém, junto com as aplicações, vêm os problemas de segurança presentes em todos os sistemas distribuídos.

Atualmente já existe uma vasta literatura sobre modelos que fornecem soluções de segurança em sistemas distribuídos, entretanto estes modelos nem sempre são adequados para a arquitetura de sistemas móveis. Neste capítulo serão apresentados conceitos fundamentais de segurança, bem como mecanismos e técnicas para obtê-la em sistemas distribuídos e em cenários de computação móvel.

2.1. Conceitos de Segurança

Existem, na literatura, diversas abordagens para determinar as características de um sistema computacional seguro. A abordagem mais aceita define um conjunto de propriedades fundamentais para um sistema seguro [21]:

- **Confidencialidade –** assegura que o sistema somente irá relevar as informações a sujeitos autorizados;
- **Integridade –** garante que as informações não são alteradas sem autorização, sejam estas alterações de caráter malicioso ou acidental;

- **Disponibilidade –** assegura que os dados de um sistema podem ser acessados por usuários legítimos sempre que solicitado.

Além destas propriedades de segurança, Landwehr [20] discute duas outras:

- **Autenticidade –** garante que cada sujeito é quem ele diz ser;
- **Não-repudiação –** assegura que um terceiro pode verificar se uma determinada transação ocorreu.

2.1.1. Políticas e Violações de Segurança

Para se manter as propriedades de segurança, apresentadas no item anterior, em um sistema computacional, é necessário definir um conjunto de regras e práticas que determinam o que é permitido e o que é proibido. Este conjunto de regras e práticas é chamado **políticas de segurança**, e são descritos em [14].

Em um sistema computacional, um **principal** é normalmente considerado a entidade (usuário, processo ou máquina) autorizada pelas políticas de segurança de um sistema. Um **intruso** é uma entidade não autorizada pela política que ganha acesso aos recursos de um sistema computacional.

As políticas de segurança definem um conjunto de regras que devem ser cumpridas. Uma **violação de segurança** ocorre quando uma destas regras é burlada. Estas violações podem ser divididas em três tipos, conforme descrito em [20]:

- **Revelação não autorizada**: surge quando a informação é revelada para entidades não autorizadas, consistindo numa violação da propriedade de confidencialidade;
- **Modificação não autorizada**: ocorre quando a propriedade de integridade é violada, seja ela devido à alteração da informação, geração ou destruição;
- **Negação de serviço**: observada quando a propriedade de disponibilidade é violada, ocasionando o não atendimento a requisições legítimas de acesso à informação.

2.1.2. Vulnerabilidades, Ameaças e Ataques

As ações maliciosas em sistemas computacionais estão ligadas aos conceitos de vulnerabilidade, ameaça e ataque [20]. Estes conceitos permitem a categorização do

ambiente, do sistema computacional, bem como de seus possíveis pontos fracos e os meios que podem ser usados por intrusos.

Vulnerabilidades de um sistema computacional são definidas como sendo pontos de fraqueza do sistema que o deixa sujeito a possíveis intrusos. Estes pontos de fraqueza podem ser falhas de procedimento, senhas mal-escolhidas e até mesmo regras configuradas ou implementadas incorretamente.

Ameaças podem ser definidas como a possibilidade ou a intenção de danificar um sistema computacional, de forma a causar efeitos indesejáveis no sistema. Uma ameaça pode ser viabilizada somente se houver vulnerabilidades no sistema.

Finalmente, **ataques** usam vulnerabilidades do sistema para concretizar um ameaça. Existem inúmeros tipos de ataques que buscam explorar as vulnerabilidades de um sistema. Estes ataques estão descritos na Tabela 1.

Tabela 1: Ataques e ameaças.

Ameaça ou ataque	Descrição
Roubo e perda	Itens contendo informações sigilosas ou de segurança crítica (como laptops, PDAs e cartões de acesso), são roubados ou perdidos.
Modificação (tampering)	O conteúdo das informações do sistema é alterado sem autorização, violando a propriedade de integridade.
Disfarce (masquerading)	A identidade de um usuário autorizado é falsificada, fornecendo direitos de acesso privilegiados.
Interceptação passiva (eavesdropping)	Transmissões de mensagem são monitoradas a fim de conseguir acesso ao conteúdo das mesmas.
Repetição (replay)	Uma mensagem enviada é armazenada, e re-transmitida para uso ilegítimo.

2.2. Modelos e Mecanismos

Os **mecanismos de segurança** em um sistema são responsáveis por implantar as políticas de segurança definidas [21]. Estes mecanismos buscam assegurar que todos os acessos a objetos são autorizados pela política definida.

Os **modelos de segurança** são usados para a definição de conceitos abstratos, propriedades e comportamentos de classes específicas de sistemas. Eles correspondem a descrições formais do comportamento de um sistema atuando segundo regras de uma

política de segurança. Estes modelos são representados na forma de um conjunto de entidades e relacionamentos [12]. Um **monitor de referência** é um modelo conceitual caracterizado por ser a entidade que recebe todas as requisições de acesso dos sujeitos autorizando ou negando o acesso de acordo com as políticas de segurança implementadas.

2.2.1. Autenticação

Em sistemas distribuídos as políticas de segurança são expressas a partir de mecanismos de autenticação e de autorização. A autenticação se refere ao processo de confirmar ou negar a identidade informada por um principal, estabelecendo sua identificação dentre um conjunto de usuários do sistema [30].

O processo de autenticação em uma comunicação necessita do que se chama de autenticação mútua dos comunicantes, onde a identidade das partes é comprovada. Outro aspecto sobre a autenticação em sistemas distribuídos é relacionado com a autenticidade nas trocas das informações que devem poder ser confirmadas para evitar ataques e ameaças como o ataque de disfarce. A resolução dos problemas de autenticidade em sistemas distribuídos envolve um serviço de autenticação como as entidades de certificação ou um serviço de autenticação como o Kerberos [41].

Segundo [17], o mecanismo de autenticação baseia-se em três classes de procedimentos:

- **Prova por conhecimento**, que consiste em estabelecer a identidade por meio de informação de conhecimento exclusivo do principal. O exemplo mais conhecido é a autenticação por meio de senha;
- **Prova por posse**, onde a identidade é confirmada através da posse de um objeto associado a uma identidade única. O exemplo mais comum são sistemas que usam *smart cards*;
- **Prova por característica**, onde a identidade é verificada por meio de característica exclusiva do principal. Esta característica é muitas vezes de origem biométrica, como por exemplo, através da avaliação da íris ou das impressões digitais.

2.2.2. Autorização

A autorização consiste na permissão ou negação do acesso a recursos do sistema. A maioria dos sistemas computacionais exige a realização do processo de autenticação para dar inicio ao processo de autorização [34].

Segundo Lampson [19], obter a fonte da requisição é chamado de autenticação, interpretar a política de acesso é chamado de autorização. Por esta razão a autenticação busca responder a pergunta "Quem disse isto?" enquanto que a autorização busca responder "Quem é confiado para acessar isto?".

Em um ambiente computacional distribuído, um dos tipos mais comuns de esquema de autorização são as listas de controle de acesso.

2.2.3. Controle de Acesso

Mecanismos de controle de acesso servem para restringir as ações que um usuário legítimo pode executar em um sistema [35]. Estes mecanismos fazem uma distinção entre o processo de autenticação e de autorização. O controle de acesso considera que a autenticação do usuário foi corretamente efetuada para então impor as políticas de segurança.

Em sistemas de controle de acesso é freqüentemente feita uma distinção entre políticas e mecanismos. **Políticas de segurança** são diretrizes de alto nível que definem a forma pela qual o acesso é controlado e como as decisões de acesso devem ser determinadas. **Mecanismos de controle de acesso** são implementações de software ou hardware que impõe as políticas de segurança [21]. A literatura trata de diversos modelos, porém três dentre eles se destacam por serem os mais empregados, sendo eles [36], [25]:

- **Controle de Acesso Discricionário (*Discretionary Access Control – DAC*):** Neste tipo de controle, é o proprietário dos dados que define quem possui direito de acesso aos dados. Entretanto se for concedida a criação de cópias dos dados, os proprietários destas cópias podem permitir o acesso de terceiros a elas, mesmo que o acesso ao conteúdo original lhes seja negado. Listas de controle de acesso e listas de competência são exemplos de mecanismos de controle de acesso que implementam as políticas de controle de acesso discricionárias. Uma descrição mais detalhada pode ser encontrada no item 2.2.4.1;

- **Controle de Acesso Mandatório (Mandatory Access Control – MAC):** Primeiramente desenvolvida para uso militar, neste tipo de modelo de controle de acesso, a informação é centralizada e distribuída de acordo com os níveis de segurança associados à informação e ao usuário ou programa que a está acessando**.** Um exemplo de implementação desta política de segurança é o modelo de Bell-LaPadula descrito em [3];
- **Controle de Acesso Baseado em Papéis *(Role-Based Access Control – RBAC)*:** Nesta classe de modelo de controle de acesso existe uma definição de papéis que são atribuídas aos usuários. Os direitos de acesso são atribuídos a papéis e não a cada usuário individualmente [25].

2.2.4. Implementação de Controle de Acesso

Para o entendimento das abordagens de implementação dos mecanismos de controle de acesso é necessário primeiramente definir o conceito de "Matriz de Controle de Acesso" proposta por [18]. Neste conceito, **sujeitos** são entidades que iniciam atividades no sistema, eles podem ser tanto usuários, como programas agindo em seu interesse. Os **objetos** do sistema são todos os recursos do sistema controlados pelo computador.

A Matriz de Controle de Acesso é, portanto, um modelo conceitual que determina os direitos que cada sujeito possui sobre cada objeto do sistema. Visualmente, é uma matriz onde as linhas indicam os sujeitos e as colunas representam os objetos do sistema, conforme indicado na Figura 1. Cada célula da matriz define os direitos de acesso do sujeito sobre o objeto.

Objetos / Sujeitos	Arquivo 1	Arquivo 2	Arquivo 3
Alice	Leitura	-	-
Bob	Leitura, escrita	Escrita	Leitura e execução
Charles	-	Leitura, escrita e execução	

Figura 1: Matriz de Controle de Acesso.

Um aspecto importante da Matriz de Controle de Acesso é que ela faz uma separação clara entre o problema de autenticação e autorização dos sujeitos.

Baseadas no conceito da Matriz de Controle de Acesso foram desenvolvidas algumas implementações de controle de acesso como as Listas de Controle de Acesso e as Listas de Competências, que são explicadas nos itens a seguir.

2.2.4.1. Listas de Controle de Acesso

As Listas de Controle de Acesso (*Access Control List* – ACL) são uma forma de implementação da Matriz de Controle de Acesso [35]. Nesta abordagem a cada objeto está associada uma ACL que relaciona os direitos de acesso de cada sujeito, como demonstrado na Figura 2.

Objetos	Listas de Controle de Acesso
Arquivo 1	Alice(Leitura), Bob(Leitura e escrita)
Arquivo 2	Bob(Leitura), Charles (Leitura, escrita e execução)
Arquivo 3	Bob(Leitura e execução)

Figura 2: Lista de Controle de Acesso.

Uma das maiores vantagens desta abordagem de implementação é a facilidade de revogação de todos os direitos de acesso a um determinado objeto. Em contrapartida, por ser focado nos objetos e não nos sujeitos, torna-se bastante difícil verificar quais são os direitos de acesso de um determinado sujeito.

2.2.4.2. Listas de Competências

Uma segunda abordagem de implementação da Matriz de Controle de Acesso é a utilização de listas de competência (*Capabilities*), que consistem e relacionar o direito de acesso de cada sujeito aos objetos do sistema [35], conforme ilustrado na Figura 3.

Ao contrário das ACLs, as listas de competência têm seu foco no sujeito, o que facilita a verificação dos direitos de acesso associados a ele, mas dificulta a modificação ou revogação dos direitos de acesso associados ao um objeto. É fácil verificar que as listas de competência são o oposto das ACLs.

Sistemas baseados em listas de competência foram desenvolvidos inicialmente na década de 70, porém não se demonstraram eficazes e comercialmente úteis. Os sistemas mais modernos têm empregado a abordagem baseada em ACLs [35].

Sujeito	Modo de acesso	Objeto
Alice	Leitura	Arquivo 1
Bob	Leitura	Arquivo 1
Bob	Escrita	Arquivo 1
Bob	Escrita	Arquivo 2
Bob	Leitura	Arquivo 3
Bob	Execução	Arquivo 3
Charles	Leitura	Arquivo 2
Charles	Escrita	Arquivo 2
Charles	Execução	Arquivo 2

Figura 3: Lista de competências.

2.2.5. Controles Criptográficos

A criptografia é a prática de obscurecer o significado de uma informação através da cifragem de forma tal que ela apenas pode ser decifrada e interpretada por pessoas para quem a informação é destinada [39].

A cifragem ocorre quando um dado sofre diversas transformações que geram uma forma alterada deste. Para fazer uma distinção entre estas duas formas da informação, existem dois termos: texto em claro (*plaintext*) e texto cifrados *(ciphertext)*.

Uma informação (ou mensagem) é chamada de texto em claro quando não sofreu nenhuma transformação no sentido de modificá-la. Após a cifragem, a informação passa a ser chamada de texto cifrado, conforme ilustrado na Figura 4.

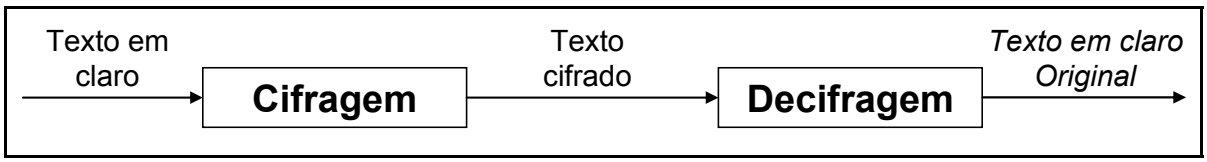

Figura 4: Codificação de Dados.

Formalmente, define-se a função de cifragem como sendo uma função E que quando aplicada sobre uma mensagem M gera um texto cifrado C.

$$E(M) = C$$

A função de decifragem *D* é tal que quando aplicada sobre *C*, ela gera a mensagem original *M*:

$$D(C) = D(E(M)) = M$$

Os algoritmos de criptografia utilizam diversas transformações matemáticas e uma informação adicional chamada "chave". Neste processo, as chaves utilizadas para cifrar e decifrar podem não ser as mesmas. Quando as chaves são iguais, o procedimento é chamado de "criptografia simétrica", e quando as chaves são diferentes, é chamado de "criptografia assimétrica".

2.2.5.1. Criptografia Simétrica

A criptografia simétrica faz uso de uma única chave para cifrar e decifrar os dados [39], conforme esquematizado na Figura 5. A confidencialidade deste tipo de criptografia depende do sigilo em relação à chave, que por esta razão também é chamada de criptografia de chave secreta.

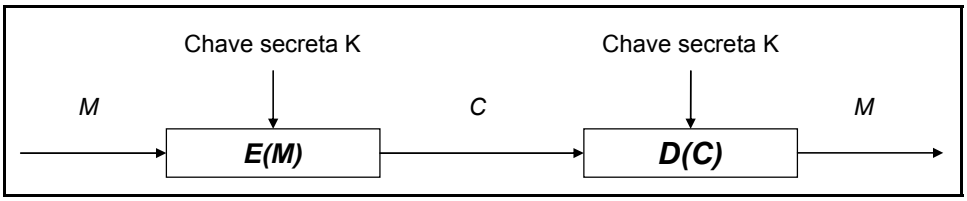

Figura 5: Criptografia de chave simétrica.

Formalmente, o esquema apresentado na Figura 5 é definido como:

$$E_k(M) = C$$

A função *D* é tal que quando aplicada sobre *C* utilizando a mesma chave *K*, ela gera a mensagem original *M*:

$$D_k(C) = D_k(E_k(M)) = M$$

Existem diversos algoritmos, como o AES [7], DES [8] e BlowFish [38], que buscam fazer as transformações matemáticas *E* e *D* de forma tal que seja computacionalmente inviável obter o valor de *M* a partir de *C* sem que seja conhecida a chave *K*, Além disso, a complexidade do cálculo necessário para obter *M* a partir de *C* cresce à medida que

aumenta o tamanho da chave *K*. Normalmente, estes algoritmos são computacionalmente eficientes, e as chaves relativamente pequenas (128 bits) oferecem níveis de complexidade suficientemente altos para serem considerados seguros.

2.2.5.2. Criptografia Assimétrica

A criptografia assimétrica se caracteriza por fazer uma transformação dos dados de acordo com um algoritmo parametrizado por um par de chaves, chamadas chaves públicas e privadas [39].

A relação entre uma chave pública e uma chave privada é tal que uma transformação criptográfica efetuada por uma chave somente pode ser revertida com o uso da outra chave, conforme exemplificado na Figura 6.

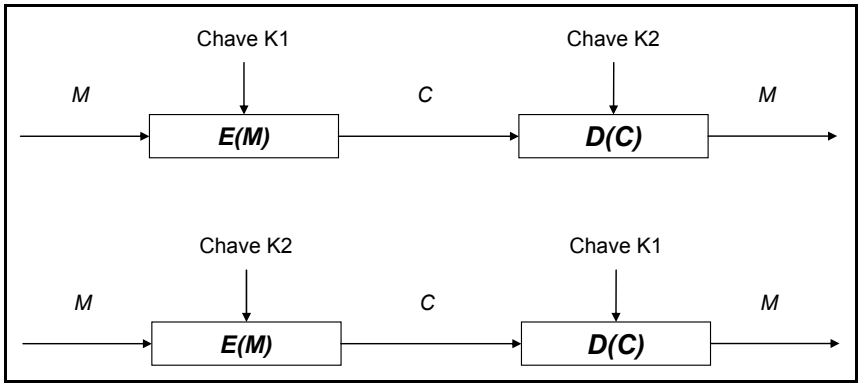

Figura 6: Criptografia de chave assimétrica.

Sendo a chave pública definida como K_1 e a chave privada como K_2, a criptografia assimétrica pode ser definida formalmente por:

$$E_{K_1}(M) = C_a \Rightarrow D_{K_2}(C_a) = D_{K_2}(E_{k1}(M)) = M$$
$$E_{K_2}(M) = C_b \Rightarrow D_{K_1}(C_b) = D_{K_1}(E_{K_2}(M)) = M$$

Apesar de K_1 e K_2 serem matematicamente relacionadas, os sistemas baseados em criptografia assimétrica são projetados de forma a ser computacionalmente inviável encontrar a K_1 a partir de K_2 e K_2 a partir de K_1. Uma conseqüência desta propriedade é a complexidade computacional geral do sistema, pois as operações matemáticas usadas necessitam de chaves relativamente longas com propriedades matemáticas específicas (por exemplo, usando números primos muito grandes) para serem considerados seguros.

2.2.5.3. Funções *one-way*

Uma função *one-way* é definida como sendo uma função *f* tal que para cada *x* no domínio de *f*, $y = f(x)$ é facilmente computada, porém dado *y*, é computacionalmente inviável encontrar *x* tal que $x = f^{-1}(y)$ [39].

Uma segunda característica de funções *one-way* é que colisões não devem ocorrer com facilidade, ou seja, deve ser improvável encontrar dois valores diferentes x_1 e x_2 tais que $f(x_1) = f(x_2)$.

Uma função *hash*, ou resumo criptográfico, é um tipo de função *one-way* que mapeia um valor de entrada de tamanho arbitrário para um valor com tamanho fixo de bits de saída. Como deve ser baixa a probabilidade de encontrar dois valores que gerem o mesmo resultado usando uma função *hash*, o valor calculado pela função é normalmente usado como uma garantia da integridade da informação.

Funções de derivação de chave *(Key Derivation Function – KDF)* também são funções *one-way*. Seu objetivo, entretanto, é tornar uma chave pequena ou senha fácil de ser quebrada através de ataques de dicionário ou de força bruta em um valor menos suscetível a estes ataques [39]. Este tipo de função produz uma **chave derivada** (CD) a partir de uma **chave básica** (CB) e outros parâmetros adicionais, como um "*salt*"[1]. A chave básica pode ser considerada uma **chave mestre**, pois a partir dela outras chaves podem ser geradas.

A definição de KDFs é abordada em diversos trabalhos e padrões, tais como SSL [49], TLS[9], PKCS#1 [31] e PKCS#5 [32]. Uma das abordagens adotada é a execução de diversas iterações sobre a chave básica: o esforço computacional necessário para derivar uma senha ainda é aceitável, enquanto que para um grande número de senhas o esforço computacional se torna inviável, dificultando muito os ataques de força bruta. Além disto, o emprego do "sal" na transformação matemática impede o cálculo prévio de um dicionário de chaves derivadas.

Este tipo de KDF pode ser representado matematicamente da seguinte forma:

[1] O "*salt*" é geralmente um valor específico para um sistema, definido previamente e de maneira aleatória.

$$CD = KDF(CB, sal, iterações)$$

Onde o parâmetro "*iterações*" se refere ao número de iterações realizadas internamente pela função.

2.3. Dispositivos móveis

Dispositivos móveis representam a evolução dos sistemas computacionais tradicionais. Usando estes dispositivos é possível interagir com outros sistemas computacionais a qualquer momento e em qualquer lugar. Efetivamente, dispositivos móveis são uma complementação dos sistemas distribuídos baseados em redes locais com fio. De uma maneira mais ampla, pode-se dizer que a computação móvel representa a eliminação das restrições de tempo e local impostas por *desktops* e redes com fio [11].

2.3.1. Caracterização

Dispositivos móveis são projetados de forma a possuírem tamanho e peso reduzidos, o que os torna altamente portáteis. De acordo com [27], dispositivos móveis de um usuário são quaisquer aparelhos portáteis pertencentes ao usuário com capacidade de processamento e armazenamento.

Seguindo esta classificação, a gama de dispositivos móveis é extensa, e inclui desde telefones celulares, com a menor capacidade de processamento, até *laptops*, capazes até de substituir *desktops* em termos de capacidade de armazenamento e processamento. Os dispositivos móveis mais difundidos atualmente, por ordem decrescente de recursos, são:

- *Personal Digital Assistant* (PDA);
- *Laptops;*
- *Smart phones.*

A computação móvel, por sua vez, pode ser definida como sendo o uso de dispositivos portáteis capazes de fazerem conexões sem fio (*wireless*) [11]. Alguns autores como Weiser [43] referenciam à computação móvel como *Ubiquitous Computing* (Ubicomp). Forman faz uma análise [11] das características e dificuldades encontradas no ambiente de computação móvel classificando-as de acordo com suas três propriedades principais: 1) comunicação sem fio; 2) mobilidade e 3) portabilidade.

Em relação à comunicação sem fio, os seguintes problemas podem ser encontrados:

- **Desconexão:** As redes sem fio sofrem de freqüentes desconexões devido a áreas sem cobertura e um alto nível de interferências e ruídos. Estas desconexões podem causar um mau funcionamento ou baixa desempenho de aplicações distribuídas, entretanto este problema pode ser contornado com o emprego de operações assíncronas;
- **Baixa taxa de transferência:** As arquiteturas desenvolvidas para dispositivos móveis devem levar em consideração a baixa taxa de transferência disponível para estes aparelhos, bem como as grandes variações que ocorrem nela devido a interferências e ruídos. Uma solução sugerida é agendar as comunicações de forma a fazer a transferência dos dados principalmente quando a conexão fornecida possuir a largura de banda desejada;
- **Redes heterogêneas:** Dispositivos móveis devem ser capazes de se comunicar em diversas redes heterogêneas com uma grande flexibilidade. A solução deste problema, segundo [22], reside em utilizar padrões voltados para a interoperabilidade.

Questões relacionadas com a mobilidade podem ser resumidas nos seguintes problemas:

- **Migração de endereços:** Devido ao deslocamento do usuário, o dispositivo utilizará diferentes pontos de acesso à rede e terá diferentes endereços. Mensagens devem ser encaminhadas ao seu endereço mais recente. Uma solução clássica para este problema seria o emprego de *broadcast*, que permite abstrair o endereço de rede;
- **Informações dependentes de localização:** Para configurar um computador, algumas informações são dependentes da sua localização espacial, e por esta razão são necessários mecanismos que permitam a obtenção destas informações de acordo com a localização do sistema.

Sistemas tradicionais como *desktops* não foram desenvolvidos para serem portáteis, por esta razão, não sofrem das mesmas restrições impostas aos dispositivos móveis. As principais diferenças em relação à portabilidade destes aparelhos são:

- **Autonomia limitada:** o consumo de energia é um aspecto importante nos dispositivos móveis, pois a bateria geralmente representa uma porcentagem

significativa do peso total. A redução das baterias torna os dispositivos mais leves e menores, mas a redução excessiva pode implicar em uma autonomia ruim, e a necessidade de recarregamento freqüente das mesmas;

- **Interface minimalista:** as restrições de tamanho impostas aos dispositivos móveis resultam em uma interface reduzida com o usuário, e muitas vezes as interfaces disponíveis em *desktops* é inadequada para estes dispositivos;
- **Pequena capacidade de armazenamento e processamento:** dispositivos móveis muitas vezes possuem uma baixa capacidade de armazenamento e processamento, o que dificulta a criação de aplicações complexas como as disponíveis em *desktops*.

2.3.2. Segurança em Dispositivos Móveis

Devido às características intrínsecas às conexões de rede sem fio, comunicações efetuadas por dispositivos móveis podem ser facilmente interceptadas e analisadas, comprometendo a segurança dos dados transmitidos [11].

A interceptação de mensagens também pode ocorrer como em sistemas distribuídos baseados em redes convencionais, e por esta razão, apesar de usarem meios diferentes, as mesmas ameaças a um sistema distribuído são observáveis em computação móvel. Desta forma, as considerações de segurança feitas para computação móvel podem ser analisadas como uma extensão das considerações feitas em sistemas distribuídos [22]. De fato, como dispositivos móveis se encontram constantemente em ambientes hostis, fora da segurança física da empresa, estes dispositivos se tornam mais suscetíveis à ameaças como roubo, perda e modificação, e podem comprometer os dados armazenados nos mesmos.

A Tabela 2 apresenta algumas ameaças e ataques que podem ocorrer em dispositivos móveis e a propriedade que enfatiza a possibilidade do ataque.

Neste trabalho, serão abordados somente os aspectos de segurança relacionados aos riscos que os dados armazenados nestes dispositivos estão expostos levando em consideração a baixa capacidade de processamento e memória destes os aparelhos.

Tabela 2: Ataques e ameaças em computação móvel.

Ataques e ameaças	Descrição	Propriedades que enfatiza a possibilidade do ataque
Roubo e perda	Itens contendo informações sigilosas ou de segurança crítica, como Laptops, cartões de acesso, são roubados ou perdidos.	Portabilidade
Modificação (*Tampering*)	O atacante altera o conteúdo das informações do sistema sem autorização, violando a propriedade de integridade.	Comunicação por rede sem fio e Portabilidade
Disfarce (*Masquerading*)	O atacante incorpora a identidade de um usuário autorizado, ganhando direito de acesso não autorizado a certos privilégio.	Comunicação por rede sem fio e Portabilidade
Interceptação passiva ou escuta (*Eaves dropping*)	O atacante monitora as transmissões de mensagem buscando conseguir acesso ao conteúdo das mensagens.	Comunicação por rede sem fio
Mensagem antiga (*replay*)	Uma mensagem enviada é armazenada e re-transmitida para uso ilegítimo	Comunicação por rede sem fio

2.4. Conclusão do capítulo

Neste capítulo foi apresentado um conjunto de definições e conceitos que caracterizam a segurança em sistemas computacionais. Em especial, discutiram-se as características de sistemas de computação móvel apresentando os aspectos de segurança que devem ser considerados nestes sistemas.

No capítulo 3 será apresentado um estudo realizado sobre a segurança dos sistemas de armazenamento de dados existentes, apresentando suas características e avaliando sua aplicabilidade em computação móvel.

Capítulo 3: Sistemas de Armazenamento Seguro

A segurança de dados é uma necessidade de crescente importância para as pessoas e organizações. Infelizmente a segurança vem freqüentemente ao custo da conveniência e conforto do usuário, desempenho ou compatibilidade com outros sistemas. A definição de um compromisso entre estes fatores é a principal fonte de pesquisa nesta área.

Neste capítulo será apresentado o estado de arte dos sistemas de armazenamento seguro de dados, ressaltando suas características, classificação, arquiteturas existentes e uma análise comparativa entre eles.

3.1. Conceitos Básicos de Segurança em Sistemas de Armazenamento

Muito do foco de trabalhos em segurança dos dados tem sido em proteger a comunicação entre clientes e servidores em uma rede não confiável. Entretanto, os dados armazenados em memórias persistentes também podem estar vulneráveis a acesso e distribuição não autorizados. Por exemplo, arquivos armazenados em um sistema de arquivo, como o do *Windows*, podem estar protegidos contra acesso não autorizado através de permissões de arquivo, contudo, se o *HardDrive* contendo estes arquivos for acessado através de um sistema como o *Linux*, as permissões de acesso são burladas permitindo a visualização dos dados dos arquivos.

Sistemas de armazenamento seguro, conforme apresentado em [30], são sistemas que implementam uma variedade de propriedades de segurança que permitem aos principais executar suas funções de maneira segura. Isto significa que os dados são armazenados de forma a garantir a confidencialidade, integridade e algumas vezes autenticidade dos dados.

Para compreender melhor os requisitos destes sistemas, é necessário apresentar conceitos importantes relacionados com sistemas de armazenamento seguro de dados como: Segurança na comunicação e em memória persistente, distribuição de chaves, revogação e granularidade da proteção. Estes conceitos são abordados nos itens a seguir.

3.1.1. Segurança de Dados em Memória Persistente e na Comunicação

Os sistemas de armazenamento de dados podem fornecer confidencialidade e integridade dos dados armazenados de duas formas: durante a transferência de dados entre o servidor de armazenamento e o cliente ou através do uso de criptografia para o armazenamento dos dados no servidor.

A segurança de dados na comunicação busca manter a integridade e confidencialidade dos dados por meio de protocolos seguros padronizados, muitas vezes são usadas soluções de *hardware* para aliviar a carga de processamento na execução de operações de criptografia no cliente ou no servidor [30].

Como descrito anteriormente, a segurança dos dados armazenados, sejam eles em disco ou em memórias removíveis, está vulnerável ao acesso e modificação não autorizados. No sentido de evitar possíveis ataques, aplica-se criptografia aos dados, a qual pode utilizar cifragem simétrica ou assimétrica, conforme descrito no item 2.2.5.

3.1.2. Gestão de Chaves

Sistemas de armazenamento seguro baseados em criptografia necessitam de chaves para a cifragem dos dados. Quando estes dados são compartilhados entre usuários, estas chaves podem ser distribuídas para os leitores e escritores de duas maneiras distintas: centralizada, através de um servidor de distribuição de chaves, ou de maneira distribuída.

O servidor de distribuição de chaves mantém as chaves que permitem a cifragem e decifragem de todos os arquivos e as listas de controle de acesso referentes a estes objetos. Para obter a chave de um determinado arquivo, o usuário deve primeiramente se autenticar e então solicitar a autorização para acesso ao arquivo que é concedida pelo servidor de grupo de acordo com as políticas de controle de acesso do sistema.

Na maneira distribuída o proprietário dos dados é responsável por fazer a distribuição das chaves para os leitores e escritores de acordo com as permissões de acesso que deseja empregar.

3.1.3. Revogação

No contexto de sistemas de armazenamento seguro, a revogação dos direitos de acesso pode ser imposta através de duas abordagens:

- Para dados cifrados somente durante a comunicação, a chave do usuário pode ser simplesmente trocada impondo os novos privilégios;
- Para dados armazenados cifrados, se faz necessário a recifragem dos dados para que os novos privilégios de acesso possam ser postos em vigor.

Na literatura [30], são definidas três classes de recifragem dos dados, de acordo com a periodicidade em que ocorrem:

- **Recifragem Agressiva**, onde o processo de recifragem ocorre imediatamente após a revogação, regravando os dados com a nova chave.
- **Recifragem Preguiçosa** *(lazy)*, que posterga a recifragem para o momento da próxima leitura ou atualização dos dados. É considerado como adequado a arquivos raramente modificados, apesar de deixarem os dados vulneráveis durante algum período;
- **Recifragem Periódica**, que oferece um compromisso entre as duas alternativas anteriores, para evitar a sobrecarga causada pela recifragem agressiva ao mesmo tempo em que limita o intervalo no qual os dados estão vulneráveis.

3.1.4. Granularidade da Proteção

A granularidade do tempo de duração das chaves usadas para cifrar e decifrar um conjunto de dados é classificada em curta e longa duração [30]: chaves de curta duração são empregadas para cifrar os dados com o intuito de protegê-los durante transmissões; chaves de longa duração são empregadas para cifrar os arquivos e podem ser compartilhadas entre vários principais.

As operações criptográficas e de gerenciamento de chaves causam uma sobrecarga no processamento de requisições em um sistema de armazenamento seguro de dados. Estes sistemas implementam diferentes estratégias buscando otimizar o processamento, como o uso de distribuição de chaves por um servidor de distribuição de chaves, e a substituição da segurança de chaves de curta duração, pois estas são modificadas a cada comunicação, pela

facilidade de gerenciamento de chaves de longa duração que são trocadas com intervalo de tempo superior às de curta duração.

Chaves de longa duração podem ser classificadas em quatro tipos:

- Chave de arquivo – usadas para cifrar um único arquivo;
- Chave de diretório – usadas para cifrar todo o conteúdo dos arquivos de um diretório;
- Chave de grupo de usuários – empregadas na cifragem de dados compartilhados por um grupo de usuários;
- Chave de grupo de arquivos – utilizadas para realizar a cifragem de um grupo de arquivos, tal como uma partição.

A utilização de chaves de curta duração reduz a vulnerabilidade, pois diminui a exposição da chave e também a quantidade de dados cifrados com a mesma chave. Contudo, chaves de longa duração são mais facilmente administradas, pois são necessárias em menor número e alteradas com menor freqüência. Além disto, o emprego de chaves de longa duração fornece proteção dos dados armazenados não apenas através de um protocolo de comunicação seguro, mas através de cifragem para o armazenamento, o que os protege de atacantes que ganham acesso ao sistema de arquivo onde se encontram.

3.1.5. Inconveniências para o Usuário

O grau de satisfação dos usuários em relação ao sistema de armazenamento seguro é fator preponderante na seleção dos mecanismos a adotar: é importante verificar o nível de inconveniência que os usuários estão dispostos a tolerar, pois quando ultrapassado, estes tendem a contornar as políticas de segurança definidas. Por exemplo, um sistema que exige trocas muito freqüentes de senhas acaba induzindo os usuários a escolher senhas fracas, porém fáceis de lembrar.

Os níveis de conveniência são definidos por [30]:

- **Sistemas Convenientes** possuem apenas uma senha para autenticação e as demais chaves são derivadas a partir dela;
- **Sistemas Inconvenientes** exigem múltiplas senhas para diferentes serviços. Estas senhas são requisitadas várias vezes e devem ser trocadas regularmente;

- **Sistemas Muito Inconvenientes** oferecem recursos de proteção com detalhamento sobre os parâmetros utilizados, como por exemplo, solicitando ao usuário que seja definidos o algoritmo e a chave que será usada para cada documento ou aplicação solicitada.

3.2. Classificação dos Sistemas de Armazenamento Seguro

O estudo da literatura demonstrou a existência de diferentes classificações dos sistemas de armazenamento seguro de dados. Dentre estes sistemas destacam-se: o *framework* de avaliação de segurança de sistemas de armazenamento de dados desenvolvido por Riedel [30], que contempla os sistemas baseados em criptografia tanto na comunicação, quanto no dispositivo de armazenamento; e o modelo de Matt Blaze apresentado em [4], que permite separar os sistemas em quatro níveis de classificação.

Neste trabalho se buscou descrever os sistemas existentes destacando as propriedades discutidas no modelo de avaliação de Riedel, porém também os classificando de acordo com critérios de Blaze. Estes modelos serão detalhados nos itens a seguir.

3.2.1. Framework de avaliação de Riedel, Kallahalla e Swaminathan

Existem várias maneiras de garantir a segurança de dados e a integridade da transferência de dados, que variam de acordo com o conjunto de ataques previstos, o nível de segurança desejado pelos proprietários dos dados, e o nível de inconveniência que os usuários estão dispostos a tolerar.

Riedel, Kallahalla e Swaminathan, em [30], apresentam um *Framework* de Avaliação de segurança dos sistemas de armazenamento de dados, onde é definida uma série de parâmetros e características para avaliação dos sistemas, sendo estes divididos em cinco categorias:

- Atores;
- Ataques;
- Primitivas de segurança;
- Granularidade de proteção;
- Inconveniência para o usuário.

As categorias apresentadas serão descritas com mais detalhes nos itens a seguir.

3.2.1.1. Atores

No modelo proposto é definida uma série de atores que executam algum tipo de ação sobre o sistema. Estes atores são definidos para efeito de análise e para limitar as ações que estes podem executar sobre um sistema:

- **Proprietários** são os responsáveis pelos dados, podendo modificá-los e destruí-los, bem como revogar ou delegar permissões de leitura e escrita a outros atores;
- **Leitores** são atores com permissão limitada apenas à leitura dos dados;
- **Escritores** são atores com permissão limitada apenas à modificação e criação de dados;
- **Canal**, que se refere ao meio de comunicação onde ocorre a transferência de dados entre os outros atores;
- **Servidores de armazenamento**, que realizam o armazenamento e transmissão dos dados de acordo com as requisições;
- **Servidores de grupos**, que são responsáveis por autenticar atores e limitar seu acesso conforme as permissões do grupo a que pertencem.

Adversários são definidos como entidades que tentam executar ações não autorizadas.

3.2.1.2. Ataques

Os ataques específicos definidos neste *framework* fazem uso de vulnerabilidades que podem ocorrer em um sistema de armazenamento, podendo levar ao comprometimento dos dados. Estes dados podem ser de curta duração, que são os dados enviados durante uma comunicação, ou dados de longa duração, que são os dados armazenados nos servidores. Desta forma, seis ameaças à segurança dos dados são definidas de acordo com o adversário que pode efetuá-las:

- **Adversário na comunicação**, como por exemplo, os ataques efetuados contra os protocolos de comunicação;
- **Adversário no servidor**, a ação de um atacante malicioso modificando um arquivo diretamente no servidor;

- **Por usuário revogado no servidor**, onde um leitor que perdeu os direitos de acesso continua a ler arquivos no servidor;
- **Adversário em conspiração com o servidor de distribuição de chaves,** que ocorre quando o servidor de distribuição de chaves auxilia o adversário para obtenção de direito de acesso aos dados do servidor de armazenamento;
- **Adversário em conspiração com leitores ou escritores,** que pode ser caracterizado pelo auxílio oferecido por leitores ou escritores a um adversário, para que este ganhe acesso aos dados armazenados.

Os ataques descritos podem ser divididos em três categorias de acordo com a violação de segurança que exercem:

- **Ataques de vazamento de informação** são os ataques onde o adversário ganha acesso a algum tipo de dado;
- **Ataques de modificação** são ataques onde o adversário faz modificações válidas nos arquivos, corrompendo a integridade dos dados sem que estas sejam percebidas pelos leitores;
- **Ataques de destruição** são ataques onde o adversário faz modificações inválidas nos arquivos corrompendo a integridade dos dados. Entretanto, ao contrário dos ataques de modificação, as alterações efetuadas podem ser verificadas pelos leitores e proprietários dos arquivos.

3.2.1.3. Primitivas de segurança

No modelo apresentado por [30], os autores consideram um conjunto integrando seis tipos de primitivas de segurança:

- Autenticação: sistemas podem autenticar seus usuários de forma **distribuída** ou **centralizada**: na forma distribuída, o proprietário dos dados autentica cada principal; na centralizada, a autenticação do principal é delegada a um servidor, como ocorre na especificação de autenticação do Kerberos [41];
- Autorização: pode ser realizada de duas formas distintas: **mediada pelo servidor,** quando este executa ações requisitadas pelos principais, ou

manipulada pelo proprietário, onde este define as permissões de acesso dos principais;

- Segurança de dados no canal de comunicação, considerando a integridade dos dados por meio da análise dos protocolos padronizados;
- Segurança de dados em disco, que analisa o tipo de criptografia utilizada na cifragem;
- Gestão de chave, classificando-a quanto ao aspecto **centralizado** ou **distribuído**;
- Revogação, que considera o período de recifragem, conforme apresentado em 3.1.3

3.2.1.4. Granularidade de proteção

Outra dimensão considerada no modelo Framework [30] é a granularidade de proteção, que considera aspectos da autorização, classificando-a em **centralizada** ou **distribuída,** e a granularidade de chaves, considerando-as de **curta** ou **longa-duração**.

3.2.1.5. Inconveniência para o usuário

Além dos aspectos relacionados à segurança e desempenho nos sistemas de armazenamento seguro de dados, o modelo de Riedel [30] contempla a avaliação da conveniência operacional dos recursos para os usuários, classificando-os conforme apresentado no tópico 3.1.5.

3.2.2. Classificação de Matt Blaze

Matt Blaze em [4] faz uma classificação dos sistemas de armazenamento seguro de dados baseado nos níveis onde são executados os controles de segurança. Neste sentido são definidos quatro níveis:

- Nível de Usuário – neste nível o usuário é responsável por coordenar o processo de cifragem dos dados, muitas vezes podendo escolher inclusive a chave empregada, bem como o algoritmo. O AxCrypt [2] e o CryptoEx [6] são exemplos destes sistemas;

- Nível de Aplicação – neste nível, os próprios programas criam seus sistemas de armazenamento seguro. Neles, o usuário possui pouco controle sobre as operações de codificação e decodificação dos arquivos, ficando a cargo do programa o controle e definição de como e quando as operações serão realizadas. O software Winzip [44], o qual solicita ao usuário a digitação de uma senha para cifrar um arquivo, é um exemplo de sistemas ao nível de aplicação;
- Nível de Sistema de Arquivos – os sistemas de armazenamento deste nível se caracterizam por estarem integrados ao sistema de arquivos do servidor de armazenamento, o que permite uma maior integração com o sistema operacional.
- Nível de Hardware – Os sistemas ao nível de hardware utilizam equipamento especializado para armazenamento seguro de dados. Existe uma gama variada de hardware que realizam esta operação, como os *smart cards* e controladores de disco, que são capazes de cifrarem todo um disco. Esta opção é geralmente rejeitada nas organizações quando existem outras alternativas, pois sua utilização implica em custos de aquisição de novo hardware.

Conforme analisado na literatura emergente [4] [5] [46], existem vários sistemas de armazenamento seguro de dados, porém serão descritos neste texto apenas os principais exemplos de sistemas ao nível de usuários e ao nível de sistema de arquivos. Esta abordagem será dada ao trabalho visto que os sistemas ao nível de hardware não são o foco de interesse e os sistemas ao nível de aplicação são soluções proprietárias de código normalmente não disponível para análise.

3.3. Armazenamento Seguro ao Nível de Usuário

Os sistemas de armazenamento ao nível de usuário, conforme descrito em [4], são uma das formas de armazenamento seguro mais simples e fáceis de implementação. A tarefa de cifragem e decifragem dos dados é responsabilidade do usuário nestes sistemas. Normalmente, a autenticação é feita através de prova por conhecimento, como por exemplo, o uso de senhas.

Diversas ferramentas desta categoria estão atualmente disponíveis: entre elas avaliaremos o AxCrypt [2], PGPDisk [29] e CryptoEx [6] nos itens a seguir.

3.3.1. AxCrypt

O AxCrypt [2] é um projeto *open source* que busca fornecer o serviço de criptografia dos dados ao nível de usuário. Este sistema emprega autenticação baseada em senha, tendo sido desenvolvido para a plataforma Windows.

Neste sistema, as chaves dos arquivos, chaves de longa duração, são derivadas da senha de autenticação e utilizadas para efetuar criptografia simétrica sobre arquivos. Este sistema, segundo o *framework* de avaliação de Riedel, pode ser classificado como muito inconveniente, pois requer que uma senha seja fornecida para cada arquivo codificado.

No AxCrypt não existe uma separação entre leitores e escritores, além disto, o compartilhamento de arquivos somente pode ser efetuado se a senha do arquivo for entregue pelo proprietário a cada ator, sendo, portanto, classificado como um sistema de gestão de chaves distribuído.

A revogação dos direitos de acesso somente pode ser feita se o arquivo tiver sua senha modificada, neste caso, uma recifragem agressiva é realizada sobre os dados. Os mecanismos criptográficos usados são criptografia simétrica usando AES [7] de 128 bits e SHA-1.

Apesar de fornecer confidencialidade e integridade dos dados, o AxCrypt é suscetível a ataques que visam a negação de serviço, pois não existe um mecanismo que impeça que um usuário do Windows apague ou danifique os arquivos.

Uma das maiores vantagens deste sistema é sua integração com o ambiente Windows (ver Figura 7): as chamadas para acessar a cifragem e decifragem de arquivos são integradas com os menus do gerenciador de arquivos do sistema operacional. Por outro lado, esta integração é uma grande desvantagem do sistema: ainda não existem versões do AxCrypt para outros sistemas operacionais.

A maior limitação do sistema é a ausência de um mecanismo para recuperação de chaves: a própria página na Internet do programa aconselha os usuários a imprimir ou copiar suas senhas por escrito, pois se a senha usada para cifrar um arquivo é perdida, o conteúdo do mesmo não pode ser recuperado.

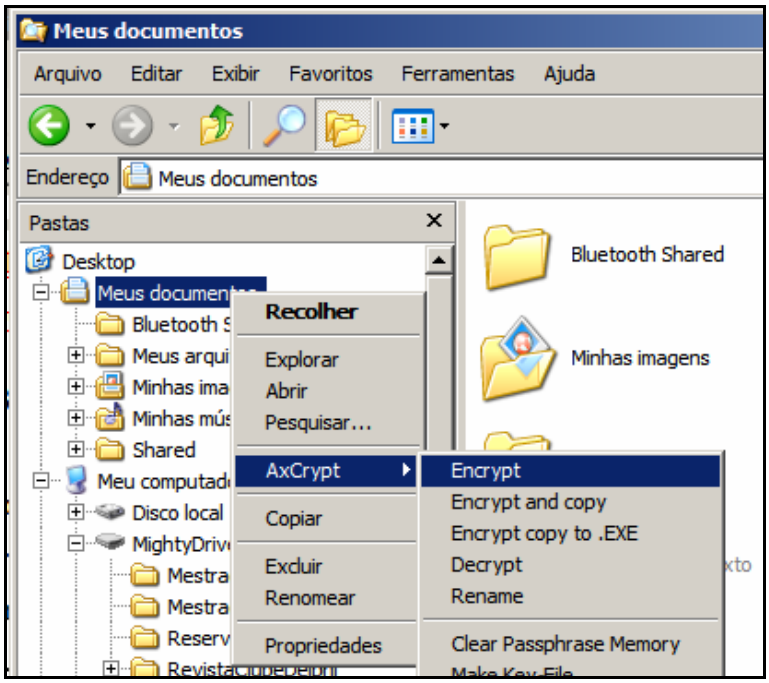

Figura 7: Integração do AxCrypt com o ambiente Windows.

3.3.2. PGPDisk e CryptoEx

O PGPDisk [29] é um sistema seguro de armazenamento de dados baseado no padrão OpenPGP [24]. Este padrão é uma evolução do PGP *(Pretty Good Privacy)* [28], que emprega criptografia assimétrica para compartilhamento de arquivos e mensagens. A escolha do algoritmo de criptografia assimétrica pode ser configurada pelo usuário.

Todos os sistemas baseados no OpenPGP usam o mesmo processo de cifragem: os arquivos são cifrados usando criptografia simétrica. A chave simétrica usada para cifrar o arquivo é gerada automaticamente pelo sistema. Esta chave simétrica, também chamada de chave do arquivo, é copiada e cifrada usando as chaves públicas de todos os usuários que poderão ter acesso ao arquivo.

Conforme descreve [30], o PGPDisk pode cifrar os dados de todo um dispositivo e armazená-los de maneira transparente, deixando a tarefa de autenticação do usuário para o sistema operacional, podendo ser classificado como conveniente.

Neste sistema não existe uma separação entre leitores e escritores. Partições virtuais são criadas para conter os dados de maneira segura: estas partições podem ser usadas para a criação de sistemas de arquivos, ou seja, a cifragem ocorre não apenas para os dados dos arquivos, mas também para a estrutura dos diretórios [26]. Em termos de granularidade de

chaves, este sistema emprega chave de longa duração, sendo ela uma chave de grupo de arquivos.

O CryptoEx [6], assim como o PGPDisk, é baseado no padrão OpenPGP. A diferença entre estes dois sistemas é a plataforma para a qual foram desenvolvidos e a forma como os dados são codificados. O PGPDisk foi desenvolvido para *desktops* e o CryptoEx foi especificamente desenvolvido para *PocketPCs*, uma classe de PDAs.

Figura 8: Gerenciamento de chaves no CryptoEx.

Do ponto de vista da autenticação, o CryptoEx funciona de maneira semelhante ao AxCrypt: o CryptoEx também delega a responsabilidade da cifragem e decifragem dos dados para o usuário, que deve escolher as chaves para codificar os arquivos, bem como definir as permissões de acesso aos arquivos cifrados, conforme ilustra a Figura 8. Este sistema é, portanto, classificado segundo o *framework* de Riedel, como muito inconveniente e com gestão de chaves distribuída. As chaves empregadas são de longa duração com uma granularidade definida por arquivo.

Não existe uma diferenciação entre leitores e escritores, por outro lado, a revogação é feita através de uma recifragem agressiva, removendo a chave do arquivo cifrada com a chave pública do usuário revogado e recifrando o arquivo com uma nova chave gerada de forma aleatória.

3.4. Armazenamento Seguro ao Nível de Sistema Operacional

Quando o sistema de armazenamento seguro está integrado com o sistema operacional, esta integração é geralmente feita nos sistemas de arquivo. Este tipo de solução normalmente possui um melhor desempenho e maior transparência, pois está disponível para todas as aplicações. Os sistemas CFS [4], TCFS [5], NCryptFS [46] e EFS [23] são exemplos deste tipo de sistema de armazenamento seguro, e são avaliados nos itens a seguir.

3.4.1. Criptographic File System

O *Cryptographic File System* (CFS) [4] foi o primeiro sistema a colocar os serviços de criptografia a nível de sistema de arquivo, tendo sido desenvolvido especialmente para sistemas operacionais Unix. Este sistema foi criado para armazenar dados tanto na máquina local quanto em servidores remotos não confiáveis.

Neste sistema, o conteúdo dos arquivos de um diretório é cifrado usando criptografia simétrica e uma chave secreta, e os dados deste diretório são então armazenados em um único arquivo no sistema de arquivos (local ou remoto). Isto permite que o sistema de arquivos trate os dados cifrados da mesma forma que os demais arquivos, permitindo inclusive *backup* dos dados cifrados de maneira transparente.

Quando o usuário realiza a cifragem, ele associa uma chave a este diretório [47]. Os arquivos ficam então disponíveis através de chamadas de sistema padrão pelo diretório "/crypt". O usuário interage com os arquivos cifrados usando chamadas de sistema padrão, sem diferenciação de arquivos que estão cifrados pelo CFS e aqueles que não estão. As demais interações são feitas através de utilitários de linha de comando.

A chave utilizada para cifrar o conteúdo dos arquivos de um diretório não é armazenada no sistema de arquivos: quando o usuário deseja cifrar ou decifrar dados, este deve colocar uma senha (*passphrase*) que será utilizada como chave. Esta senha deve possuir no mínimo 16 caracteres, pois é o comprimento necessário para o algoritmo de criptografia usado, o DES [8]. Como a chave não é armazenada no sistema de arquivos, é possível armazenar os dados em servidores não confiáveis.

Conforme a classificação de Riedel, este sistema é inconveniente, pois solicita senhas muito longas para que o usuário possa acessar seus dados, e para cada diretório codificado

com uma chave diferente, é necessário fornecer um outra senha. A chave empregada é de longa duração, sendo sua granularidade de grupo de arquivos.

Este sistema não prevê o compartilhamento de arquivos [5]: ou seja, cada usuário deve receber a *passphrase* para obter acesso aos arquivos. Neste sentido, o CFS se classifica como um sistema com distribuição de chaves distribuída.

3.4.2. Transparent Cryptographic File System

O *Transparent Cryptographic File System* (TCFS) [5] foi especificado baseando-se no CFS, porém com diversas melhorias que facilitam seu uso, funcionando de forma transparente aos usuários.

A arquitetura do TCFS é baseada na premissa que os dados devem ser armazenados cifrados no sistema de arquivos sem que este tenha acesso às chaves necessárias para decifrar os dados. Isto permite que, por exemplo, os dados sejam armazenados em servidores não confiáveis.

Este sistema de armazenamento seguro distribuído foi implementado para o Unix como uma forma modificada do cliente NFS [46]. Toda vez que uma aplicação rodando do lado do cliente precisa ler dados, o cliente NFS modificado requisita o bloco de dados necessário ao servidor. O servidor envia os dados de forma cifrada e o cliente NFS modificado decifra os dados antes de devolvê-los para a aplicação. A operação de escrita ocorre de forma semelhante: antes de enviar os dados para o servidor, o cliente NFS modificado cifra os dados.

3.4.2.1. Codificação e Decodificação dos dados

O TCFS permite que cada arquivo seja cifrado usando algoritmos diferentes de criptografia, pois cada arquivo possui um cabeçalho (*header*) que contém informações sobre o arquivo, e inclusive qual cifra foi utilizada. Normalmente a cifra empregada é simétrica, e a chave usada para cifrar cada arquivo é gerada de forma aleatória. Esta chave é cifrada com a chave mestre do usuário e armazenada no cabeçalho do arquivo.

No sentido de fazer uma diminuição da granulariade do arquivo e aumentar o desempenho do processo de cifragem e decifragem dos arquivos, faz-se uma separação dos dados em blocos. O esquema para cifrar os blocos de dados de cada arquivo é ilustrado na Figura 9: cada bloco de dados é cifrado com uma chave diferente (bk). Esta chave é obtida

calculando o *hash* da chave do arquivo (fk) juntamente com o número do bloco. A chave do bloco é usada com a cifra escolhida para cifrar o bloco no modo CBC [8]. Para garantir a integridade do bloco, é calculado o *hash* da concatenação da chave do bloco com o conteúdo do bloco. Nenhuma medida é tomada no sentido de garantir a integridade do arquivo como um todo, apenas a integridade dos blocos é garantida.

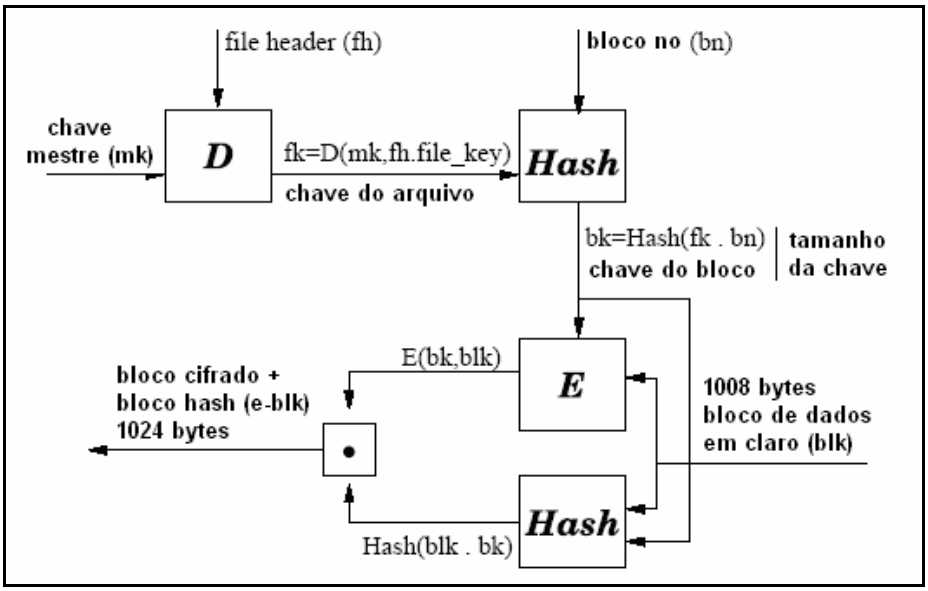

Figura 9: Criptografia de blocos de dados no TCFS.

3.4.2.2. Gerenciamento de Chaves no TCFS

O TCFS separa o gerenciamento de chaves do resto do sistema armazenamento seguro. O TCFS emprega chaves de longa duração por arquivo. Esta separação pode ser observada pelo uso de uma interface simples e independente para o fornecimento das chaves para o cliente NFS modificado. Usando esta interface, sistemas de gerenciamento de chaves mais complexos podem ser implementados. O próprio TCFS oferece quatro serviços de gerenciamento de chaves que servem a diferentes propósitos, que são:

- **Gerenciamento de chave *Raw* (RKM)**, que é o serviço mais básico implementado. Neste tipo de gerenciamento a API do RKM fica disponível para as aplicações do usuário que vão utilizá-la para passar a chave ao cliente NFS. Esta chave será usada para cifrar e decifrar os dados. Este tipo de gerenciamento de chaves é classificado por Riedel como inconveniente, pois em última instância implica solicitar informações ao usuário para cada arquivo acessado;

- G**erenciamento de chave Básico (BKMS)**, onde a chave usada para cifrar os arquivos é cifrada com a senha de *login* do usuário no banco de dados de chaves. Para acessar os arquivos cifrados o usuário extrai sua chave mestre do banco de dados de chaves e a passa para a camada do TCFS. Este sistema é classificado como conveniente, pois somente uma senha é solicitada ao usuário.
- **Gerenciamento de chaves *Group Sharing***, é utilizado para o compartilhamento de arquivos entre usuários. Neste tipo de gerenciamento de chave, os arquivos se tornam disponíveis para o grupo se um número mínimo de membros do grupo estiverem "ativos" no sistema. Para alcançar este objetivo uma chave é gerada para o grupo e é distribuída entre seus membros de forma tal que só pode ser reconstituída se pelo menos um número mínimo deles fornecerem sua parte da chave. A classificação do grau de conveniência deste sistema de acordo com o *framework* de Riedel se torna, difusa, pois se for levado em consideração cada usuário individualmente, o sistema é conveniente, no entanto se for considerado o grupo de usuários, este sistema se torna inconveniente, pois é necessário que um número mínimo de senhas sejam fornecidas para que os arquivos possam ser acessados.
- **Gerenciamento de chaves Kerberos**, que propõe uma arquitetura integrada com o serviço de autenticação distribuído Kerberos [41]. Nesta modalidade de gerenciamento um novo componente é adicionado, o *TCFS Key Server* (TCFSKS), que mantém um banco de dados das chaves mestres. Para acessar uma chave, o cliente NFS modificado se autentica com o servidor Kerberos para receber um *ticket* e uma chave para comunicação com o servidor de chaves. Usando o seu *ticket*, o cliente NFS modificado se comunica com o TCFSKS para obter a chave mestre. As operações de leitura e escrita ocorrem da mesma forma que na modalidade Básica, com a exceção da forma como as chaves são obtidas. Neste gerenciamento de chaves, a distribuição de chaves é considerada centralizada pela classificação de Riedel.

Todos estes sistemas de gerenciamento de chaves são baseados em cifras simétricas. Entretanto, dada a modularidade do TCFS, é possível implementar um sistema de gerenciamento de chaves usando criptografia assimétrica.

3.4.3. CryptFS e NCryptFS

NCryptFS [46] é um sistema de armazenamento seguro que permite que dados sejam protegidos tanto localmente como em servidores remotos.

O CryptFS [48] foi desenvolvido como protótipo para testar as características de um sistema de armazenamento seguro integrado com a arquitetura de sistemas de arquivos em sistemas operacionais Unix. O NCryptFS é o sucessor do CryptFS, implementado a partir dos experimentos feitos e resultados obtidos com o CryptFS.

Ambos os sistemas foram desenvolvidos para operar intimamente ligados ao kernel do sistema operacional. Por esta razão, estes são capazes de suportar sistemas de arquivos que sejam suportados pelo sistema operacional. Esta integração com o kernel também permite um bom desempenho, mas ao mesmo tempo torna ambos os sistemas bastante especializados para o sistema operacional escolhido.

Ao contrário do CryptFS, o NCryptFS suporta diversas cifras e permite um certo compartilhamento de arquivos, definindo quatro tipos de usuários: administrador do sistema, proprietário do arquivo, escritores e leitores. O administrador do sistema é responsável por instalar e montar o NCryptFS. O proprietário do arquivo é quem possui a chave do arquivo e pode delegar o direito de acesso a outros usuários. Os escritores e leitores não possuem acesso à chave do arquivo, e possuem somente os direitos que lhes são delegados. Do ponto de vista da classificação de Riedel, neste sistema existe uma separação clara entre leitores, escritores e proprietários do arquivo.

Assim como no CFS, o NCryptFS associa chaves a diretórios para o processo de cifragem e decifragem do conteúdo dos arquivos do diretório. O proprietário da chave do diretório define os direitos de acesso de outros usuários. Antes que os dados possam ser usados por todos os usuários autorizados, o proprietário do arquivo deve passar a chave do diretório (uma *passphrase*) para o NCryptFS. A *passphrase* é derivada usando a função PKDF2 definida em PKCS#5 [32], sendo a chave derivada usada para cifrar e decifrar os dados dos arquivos. Para que os leitores e escritores possam usar os arquivos, esta chave derivada fica armazenada em memória volátil durante o período em que o proprietário está ativo.

De acordo com a classificação de Riedel, este sistema possui uma granularidade de chave de grupo de arquivos, sendo esta chave de longa duração. Devido à forma como as

permissões de acesso ao arquivo são feitas considera-se o sistema de distribuição de chaves distribuído. Além disto, o sistema pode ser considerado inconveniente, pois é necessário fornecer uma senha diferente para cada diretório cifrado com uma chave diferente.

Os autores do CryptFS, em seu artigo, apresentam uma comparação de desempenho entre o TCFS e o CryptFS, demonstrando que o CryptFS é mais rápido que o TCFS. Entretanto, os autores do TCFS em [5] argumentam que o CryptFS não faz uma verificação da integridade dos arquivos, o que de fato torna o processo de cifragem e decifragem dos dados mais simples, ao custo da inserção de vulnerabilidades na arquitetura do sistema.

3.4.4. Encrypting File System

O Encryption File System (EFS) [23] é um serviço de armazenamento seguro de arquivos desenvolvido pela Microsoft para os sistemas operacionais Windows 2000, Windows XP e Windows Server 2003. Ele permite que os dados sejam armazenados de forma segura localmente ou em servidores remotos.

Por ser intimamente integrado ao sistema de arquivos adotado pela Microsoft (NTFS), a cifragem e decifragem de arquivos ocorrem de maneira transparente para o usuário. Quando um usuário abre um arquivo, este é decifrado pelo EFS à medida que os dados são lidos do disco. Quando dados são salvos, o EFS os cifra à medida que são escritos para o disco. Para cifragem dos dados, o EFS utiliza a cifra DESX[2] [23]. É possível, no Windows XP, configurar o sistema para usar o 3DES [8].

O EFS permite que arquivos protegidos sejam compartilhados entre usuários sem que a chave do arquivo tenha que ser passada diretamente para cada usuário com direito de acesso ao arquivo. Para viabilizar esta funcionalidade, o EFS utiliza um sistema de gerenciamento de chaves semelhante ao PGP *(Pretty Good Privacy)* [28] que combina criptografia assimétrica e simétrica.

No EFS a cifragem é feita ao nível de arquivo, mas usuários podem escolher cifrar todos os arquivos de um diretório. Quando isto ocorre, o EFS automaticamente cifra todos os arquivos inseridos dentro no diretório.

[2] O DESX é um algoritmo proprietário inventado pela Microsoft, e consiste em uma variação do algoritmo DES.

Para evitar a perda de dados devido à perda de senhas por parte dos usuários, o EFS define os *Data Recovery Agents* (DRA) que podem recuperar os dados nestas situações. Os DRA são definidos pelo proprietário do arquivo, e sua implementação segue os moldes apresentados pelo OpenPGP para compartilhamento de arquivos.

3.4.4.1. Gerenciamento de Chaves

No EFS cada usuário possui um par de chaves pública e privada. A chave pública é armazena em texto em claro, pois deverá ser acessada por outros usuários. A chave privada é cifrada com uma chave simétrica de 64 bits que é a chave mestre do usuário.

A chave mestre do usuário é cifrada com uma derivação da senha do usuário. Somente após fazer o login no Windows a chave privada do usuário fica disponível. Quando o usuário troca sua senha, a chave mestre é cifrada novamente usando uma chave derivada da nova senha.

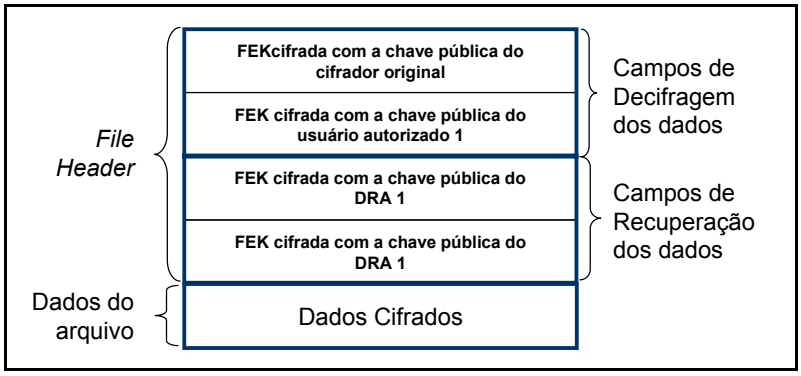

Figura 10: Formato de um arquivo cifrado no EFS.

Cada arquivo cifrado possui um cabeçalho (*file header*) que contém a informação necessária para a cifragem e decifragem do arquivo. Para cada arquivo é gerada uma chave simétrica única, chamada *File Encryption Key* (FEK). Esta chave é usada juntamente com uma cifra simétrica para cifrar o arquivo. A FEK é então cifrada com a chave pública do usuário e armazenada com o arquivo, no seu *File Header*. Quando o arquivo é compartilhado, a FEK é novamente cifrada com a chave pública dos demais usuários que também possuem direito de acesso ao arquivo. As diferentes cópias cifradas da FEK são armazenados no *File Header*, como mostra a Figura 10. Se algum DRA foi definido, a FEK deve também ser cifrada usando a chave pública correspondente desta entidade. As cópias da FEK cifradas com as chaves públicas dos DRAs também são armazenados no *File Header*.

Segundo a classificação de Riedel, a chave empregada para criptografia dos dados possui uma granularidade de chave de arquivo sendo esta de longa duração. No que diz respeito à interação com o usuário, o EFS é considerado conveniente, pois a mesma senha do *login* no sistema operacional é reutilizada de forma a gerar as chaves necessárias para a codificação dos arquivos.

3.4.5. CryptoCache

O CryptoCache [37] é um sistema de armazenamento seguro de dados, voltado para o uso em dispositivos móveis, que visa armazenar os dados em um servidor não confiável para que o dispositivo possa recuperar os dados a medida que for necessário.

O autor argumenta que, por questões de custo, um dispositivo móvel poderia se conectar a um servidor de arquivo que esteja próximo à sua localidade ao invés de se conectar diretamente ao seu *home server*, conforme ilustrado na Figura 11.

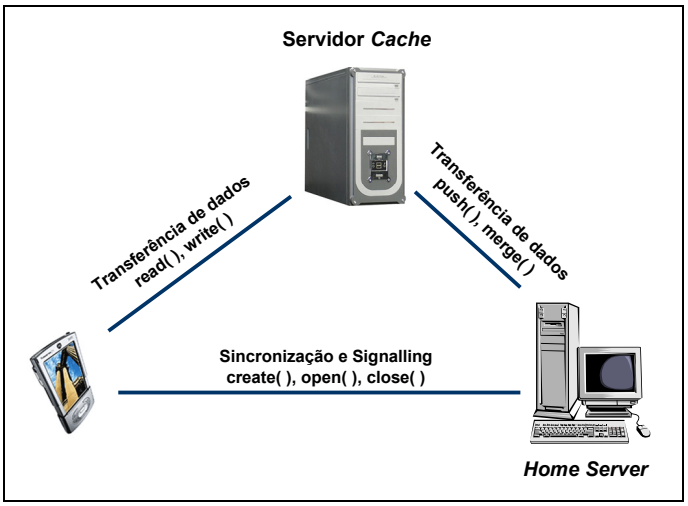

Figura 11: Arquitetura do CryptoCache

Este sistema se baseia na arquitetura do CFS, TCF e CryptFS, entretanto se preocupa com a questão do compartilhamento de arquivos entre usuários, fazendo a distinção entre leitores e escritores. Buscando impor esta separação o CryptoCache emprega chaves assimétricas para a cifragem dos arquivos. Um par de chaves é gerado, sendo a chave pública entregue aos leitores e a privada aos escritores.

Todos os arquivos são cifrados pelo *home server* e somente são decifrados pelo cliente. Fornecendo confidencialidade, integridade e autenticidade para os dados armazenados.

O autor do artigo ressalta que neste sistema se assume um modelo de confiança onde os leitores e escritores são confiáveis e, portanto não irão entregar sua chave para outro ator ou adversário. Um outro aspecto que é mencionado no artigo é que um *hardware* adicional auxiliaria a realizar as operações criptográficas dado que criptografia assimétrica é empregada.

O acesso aos dados é feito da mesma forma que no CFS, sendo, portanto considerado um sistema inconveniente de acordo com a classificação de Riedel. Outra constatação feita é que o CryptoCache possui uma gestão de chaves distribuída, pois o proprietário do arquivo passa as chaves do arquivo para os demais leitores e escritores. A granularidade das chaves é feita por grupo de arquivos, assim como no CFS.

3.5. Considerações

A avaliação destes sistemas de armazenamento seguro nos permite concluir que a criptografia ao nível de usuário traz consigo flexibilidade, pois nesta modalidade é possível definir exatamente o que deve ser cifrado e de que forma. Porém, ao mesmo tempo em que oferece mais opções para o usuário, exige maior conhecimento e cautela para evitar que dados sensíveis permaneçam desprotegidos ou que sejam perdidos por descuido no gerenciamento das chaves. Ou seja, esta modalidade de armazenamento seguro é muito vulnerável a erros do usuário.

O armazenamento seguro ao nível de aplicação permite que aplicações compartilhem dados somente se estas implementarem o mesmo sistema de armazenamento seguro. A maior vulnerabilidade destes sistemas é a qualidade das implementações: basta uma aplicação ser suscetível a um ataque para que a segurança dos dados compartilhados seja comprometida.

Sistemas de armazenamento seguro aos níveis de usuário, aplicação, sistema operacional e *hardware* descritos nos itens anteriores apresentam recursos diferenciados, que determinam sua adequação aos ambientes computacionais e necessidades dos usuários. A tabela abaixo sintetiza as propriedades dos sistemas avaliados:

Tabela 3: Tabela comparativa de sistemas de armazenamento seguro.

Propriedades \ Sistemas		CFS	TCFS	EFS	CryptFS	NCryptFS	PGPDisk	CryptoEX	CryptoCache
Níveis de Matt Blaze	Usuário						X	X	
	Sistema de Arquivos	X	X	X	X	X			X
Framework	Atores	Leitores e escritores não são diferenciados	Leitores e escritores não são diferenciados	Leitores e escritores não são diferenciados	Leitores e escritores não são diferenciados	Separação entre leitores e escritore	Leitores e escritores não são diferenciados	Leitores e escritores não são diferenciados	Separação entre leitores e escritore
	Primitivas de segurança;	Distribuição de chaves, autenticação e autorização distribuídas.	Varia de acordo com o gerenciamento de chaves empregado	Distribuição de chaves, autenticação e autorização distribuídas.	Distribuição de chaves, autenticação e autorização distribuídas.	Distribuição de chaves, autenticação e autorização distribuídas.	Distribuição de chaves, autenticação e autorização distribuídas.	Distribuição de chaves, autenticação e autorização distribuídas.	Distribuição de chaves, autenticação e autorização distribuídas.
	Granularidade da Proteção	Criptografia simétrica com chaves de longa duração	Criptografia simétrica com chaves de longa duração	Criptografia simétrica e assimétrica com chaves de longa durção	Criptografia simétrica com chaves de longa duração	Criptografia simétrica com chaves de longa duração	Criptografia simétrica e assimétrica com chaves de longa durção	Criptografia simétrica e assimétrica com chaves de longa durção	Criptografia assimétrica com chaves de curta e longa duração
	Inconveniência para o usuário	Inconveniente	Varia de acordo com o gerenciamento de chaves empregado	Conveniente	Inconveniente	Inconveniente	Conveniente	Muito Inconveniente	Inconveniente
Outras	Independência de plataforma	Não	Não	Não	Não	Não	Não	Não	Não
	Recuperação de chaves	Não	Não	Sim	Não	Não	Não	Não	Não

É possível observar, a partir da Tabela 3, que os sistemas atuais de armazenamento seguro analisados são satisfatórios para os mais diversos usos possíveis em *desktops*. Entretanto, se avaliados dando ênfase para aplicações para PDAs, estes sistemas não são adequados às restrições de processamento, memória, capacidade de armazenamento, funcionalidade do sistema operacional e número de usuários ativos simultaneamente.

Mais especificamente, os sistemas EFS, CryptoEx, CryptoCache e PGPDisk empregam criptografia assimétrica para permitir o compartilhamento de arquivos entre usuários. Esta abordagem não é adequada à baixa capacidade de processamento dos dispositivos móveis, pois algoritmos de criptografia assimétrica requerem muita potência de cálculo. Ainda, somente o EFS fornece um serviço para a recuperação dos dados do usuário.

Por natureza, PDAs são dispositivos mono-usuário. Por esta razão, o TCFS também é inadequado para o uso em PDAs, dado que o compartilhamento de arquivos depende de um sistema de gerenciamento de chaves que exige mais de um usuário ativo simultaneamente no sistema.

Sistemas como o CFS, CryptFS, NCryptFS e AxCrypt sofrem de problemas de usabilidade: a falta de sistemas de gerenciamento de chaves ou a necessidade de uso de chaves extremamente longas podem levar à perda de dados ou, simplesmente, ao abandono do sistema de armazenamento seguro.

De certa maneira, é possível simplificar a comparação dos diferentes sistemas a um compromisso entre confidencialidade, integridade, desempenho e usabilidade. Este compromisso é ainda mais limitado em dispositivos móveis:

- Todos os sistemas tentam manter a confidencialidade dos dados, porém alguns não mantêm a integridade;
- Os sistemas gerenciamento de chaves atuais não oferecem desempenho e usabilidade ao mesmo tempo. Sistemas que priorizam o desempenho geralmente adotam soluções de pouca usabilidade (muitas senhas, senhas longas), enquanto que sistemas com maior usabilidade usam criptografia assimétrica, que causa um impacto significativo no desempenho.

Entre os sistemas atuais, somente o EFS oferece a capacidade de recuperação dos arquivos caso o usuário esqueça sua senha. Este serviço requer, entretanto, que a chave de acesso aos arquivos seja confiada a outro usuário, além do proprietário dos mesmos.

3.6. Conclusões

Neste capítulo foram apresentados diversos sistemas de armazenamento seguro, tanto de origem acadêmica como comercial. Apesar das características oferecidas por estes sistemas ser suficiente para os usos feitos em *desktops*, as premissas destes sistemas não podem ser transportadas para dispositivos móveis, principalmente por causa das restrições de processamento, memória, armazenamento e funcionalidade do sistema operacional. Foi apresentada uma tabela comparativa reunindo o estado da arte em sistemas de armazenamento seguro que permite avaliar as escolhas feitas na definição destes sistemas.

Capítulo 4: *MobSec* - Armazenamento Seguro em Dispositivos Móveis

O crescimento substancial do uso de dispositivos móveis para aplicações empresariais, associado à facilidade e rapidez de acesso à informação, oportunizou um aumento exponencial nos riscos e ameaças a que estes aparelhos estão sujeitos.

Os dispositivos móveis estão freqüentemente sujeitos à ameaças como perda e roubo que poderiam, em última análise, no mínimo, comprometer o sigilo de informações.

Neste capítulo serão abordados os aspectos estruturais e comportamentais do modelo proposto para o armazenamento seguro de dados em dispositivos móveis que visa trazer soluções para as ameaças citadas.

4.1. O Projeto MobSec

As atuais arquiteturas disponíveis para prover o armazenamento seguro de dados, descritas e analisadas no capítulo 3, caracterizam-se pela ampla dependência e interligação com o sistema operacional além de não apresentarem, num mesmo sistema, confiabilidade e bons níveis de usabilidade e desempenho quando portados para dispositivos móveis.

Estas limitações e a falta de uma padronização para armazenamento seguro em dispositivos móveis motivou o desenvolvimento do projeto *Mobile Secure Storage* (MobSec), que consiste num conjunto de serviços para armazenamento seguro de dados em dispositivos móveis.

A concepção do projeto MobSec teve por objetivo definir uma arquitetura capaz de fornecer um serviço de armazenamento de dados com boas propriedades de usabilidade, desempenho e segurança para dispositivos móveis.

4.2. Aspectos da Arquitetura MobSec

A arquitetura do MobSec do lado do cliente é destinada a fornecer confidencialidade e integridade dos dados armazenados em dispositivos móveis. Estes dados são apresentados apenas a usuários autorizados assegurando estas propriedades através de mecanismos criptográficos e de controle de acesso.

Nesta arquitetura, quando um dispositivo é roubado ou perdido, é possível recuperar a parte dos dados já enviada para um servidor remoto para armazenamento de backup e processamento. A operação de envio de dados ao servidor remoto é citado no texto por sincronização entre o dispositivo móvel e o servidor (servidor de sincronização).

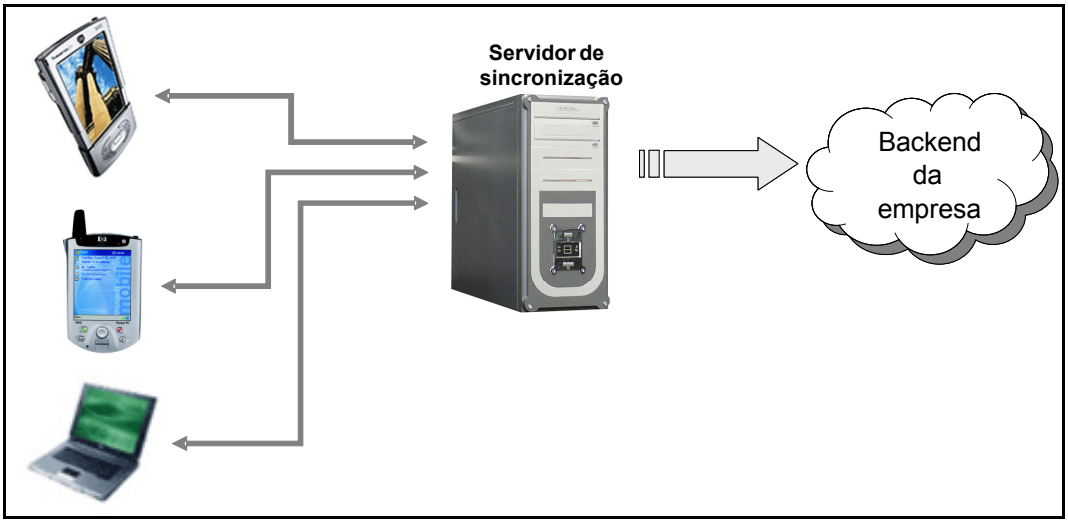

Figura 12: Cenário B2E utilizando dispositivos móveis.

O serviço de armazenamento seguro fornecido pelo MobSec pode servir a um cenário B2E onde funcionários usam estes aparelhos para coletar dados, como por exemplo, pedidos de clientes, para posteriormente transferí-los para o servidor de sincronização da empresa, conforme ilustra a Figura 12. Após a sincronização, estes dados são passados para o *Backend* da empresa onde serão processados, encaminhando, por exemplo, os pedidos dos clientes para a linha de produção.

Por se tratar de um cenário *offline*, os dados devem ficar protegidos no dispositivo móvel até que estejam seguros no servidor, o que ocorre após a sincronização. Para esta finalidade, um sistema de gerenciamento de chaves foi desenvolvido baseado no estudo feito na literatura [24] [40] [46].

Figura 13: Arquitetura do MobSec.

Devido à natureza distribuída do sistema, a arquitetura do MobSec foi concebida em dois módulos: Cliente e Servidor, conforme esquematizado na Figura 13.

O Módulo Cliente, residente no dispositivo móvel, apresenta arquitetura que permite proteger os dados contra acesso ou modificação não autorizados. Este módulo pode ser acoplado à uma plataforma que provê serviços para as aplicações do usuário.

Por sua vez, o Módulo Servidor deve ser capaz de receber os dados enviados nos períodos de sincronização e interpretá-los para processamento.

4.2.1. Gerenciamento de chaves

No projeto MobSec procurou-se desenvolver um sistema de gerenciamento de chaves de forma a contemplar os requisitos de usabilidade e desempenho, imprescindíveis num ambiente de computação móvel.

Para alcançar este objetivo empregou-se somente criptografia de chave simétrica, por se tratarem de algoritmos mais eficientes quando comparados aos de chave assimétrica. Entretanto, o uso de criptografia de chave simétrica para compartilhamento seguro de arquivos em grupos de usuários gera alguns desafios relativos ao gerenciamento de chaves.

Para compartilhar um arquivo cifrado usando criptografia simétrica, é necessário que a chave necessária para a decodificação dos dados esteja disponível a todos os usuários

autorizados. Na literatura existem três principais abordagens que buscam solucionar este problema:

- Emprego de criptografia de chave assimétrica para codificar a chave simétrica do arquivo. Esta abordagem pode ser observada em sistemas derivados do OpenPGP [24] e no EFS, conforme descrito no item 3.4.4;
- Emprego de conceito de limiar (*Threshold*) [40] para a geração da chave simétrica do arquivo somente quando um número mínimo de membros do grupo estiver ativo. Este conceito é empregado no TCFS, conforme descrito no item 3.4.2;
- Liberação da chave simétrica do arquivo para os membros do grupo quando o proprietário estiver ativo no sistema. Este artifício é usado pelo NCryptFS, descrito no item 3.4.3.

Estas abordagens exigem mais de um usuário ativo no sistema ou necessitam de muita potência de cálculo para o emprego de criptografia assimétrica. Por estas razões, não se ajustam às limitações existentes no cenário da computação móvel.

Para solucionar este problema, a arquitetura MobSec propõe um sistema de gerenciamento de chaves capaz de fornecer a infra-estrutura de compartilhamento seguro de arquivos, baseado somente em criptografia simétrica e funções de derivação de chave além de atender as restrições de um cenário mono-usuário também consideram as limitações dos dispositivos móveis.

O gerenciamento de chaves foi separado em dois serviços:

- Serviço de Gerenciamento de Chaves do Cliente;
- Serviço de Gerenciamento de Chaves do Servidor.

Nesta arquitetura as chaves simétricas necessárias para cifrar e decifrar os arquivos de um determinado grupo ou de um usuário são geradas no servidor e transferidas para os dispositivos móveis. Estas chaves são chamadas de chave mestre de grupo e chave mestre de usuário respectivamente.

4.2.1.1. Serviço de Gerenciamento de Chaves do Cliente

O Serviço de Gerenciamento de Chaves do Cliente (SGCC) se encarrega de armazenar as chaves mestres e recuperá-las quando requisitado. Cada usuário possui seu próprio *Keystore* onde se encontram todas as chaves mestres necessárias para executar as operações de codificação e decodificação dos arquivos protegidos.

Figura 14: Gerenciamento de chaves no SGCC.

Para garantir que o *Keystore* é desbloqueado apenas quando o usuário se autentica corretamente, o mecanismo de bloqueio e desbloqueio utiliza uma cifra simétrica juntamente com uma chave simétrica gerada a partir de informações fornecidas durante o processo de autenticação. Este mecanismo é ilustrado na Figura 14.

Quando o usuário se autentica, o *Keystore* é decodificado permitindo que as chaves armazenadas sejam acessadas. Para proteção das chaves mestres, as referências para o *Keystore* decifrado são removidas da memória quando o usuário está inativo.

As chaves mestres armazenadas no *Keystore* do usuário são chaves simétricas usadas para gerar as chaves que irão de fato cifrar os arquivos. São elas:

- Chaves mestres dos grupos aos quais o usuário pertence;
- Chaves mestres do usuário.

Para se obter uma arquitetura com maior flexibilidade, o sistema foi desenvolvido de forma a suportar diversas versões de chaves para recodificar os arquivos protegidos numa situação onde uma chave mestre é comprometida ou trocada. As chaves mestres do usuário versão 1.0 e 2.0 da Figura 14 exemplificam o sistema de versões.

Visando permitir o compartilhamento de arquivos entre membros de um mesmo grupo, o conteúdo de uma versão da chave mestre de um grupo é o mesmo para todos os membros deste; entretanto, o conteúdo de uma chave mestre de usuário é único. Para impedir acesso não autorizado, os arquivos de grupo e de usuário são cifrados com uma derivação da chave mestre do grupo e do usuário respectivamente.

4.2.1.2. Serviço de Gerenciamento de Chaves do Servidor

O Serviço de Gerenciamento de Chaves do Servidor (SGCS) proposto no MobSec, tem por missão realizar o gerenciamento de chaves no que compete à geração destas para armazenamento e recuperação de arquivos cifrados nos dispositivos móveis.

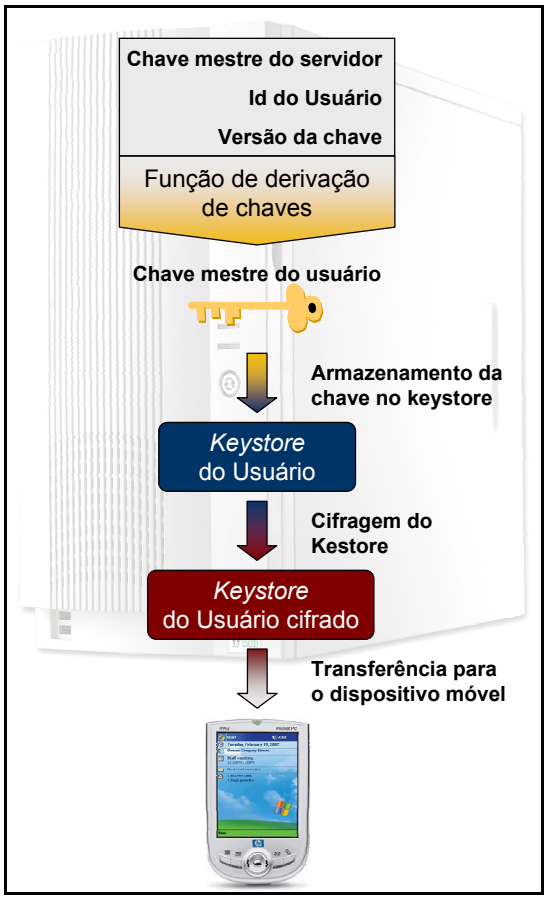

Figura 15: Gerenciamento de chaves no SGCS.

Todas as chaves usadas para cifrar os arquivos nos dispositivos móveis, são geradas no servidor e transferidas para os aparelhos durante o período de sincronização, conforme ilustrado na Figura 15. Este procedimento garante que o servidor poderá sempre permitir a recuperação dos dados contidos nos dispositivos móveis.

A geração das chaves somente deve ser possível no servidor, e para tal utiliza-se um segredo para criá-las chamado de chave do servidor. Esta chave também deve ser armazenada de maneira segura. De acordo com os requisitos de segurança, um hardware especializado, como um *smartcard*, pode ser empregado para realizar o armazenamento desta chave.

Um mecanismo de limiar (*Threshold*) [40] pode, por exemplo, ser usado para definir um numero mínimo de membros (administradores) que devem fornecer as suas partes do segredo para gerar uma chave, que seria usada para aumentar a segurança do armazenamento da chave do servidor.

Cada usuário recebe somente as chaves mestres de usuário e de grupos associadas aos seus direitos de acesso. Após a geração as chaves são armazenadas em um *Keystore* do usuário e protegidas através de uma derivação do login e senha do mesmo utilizada no dispositivo móvel.

Os *Keystores* dos usuários são transferidos para os aparelhos através do carregamento de mídia removível ou através de um protocolo de comunicação segura como o SSL [49] com autenticação mútua do Cliente e Servidor.

Esta abordagem traz a vantagem de poder recuperar os dados armazenados no dispositivo móvel mesmo que o usuário tenha esquecido a sua senha, ou que por alguma razão não faça mais parte da empresa, pois a chave poderá ser recuperada no servidor e armazenada em um novo *Keystore*.

4.2.2. Módulo Cliente

A estrutura do Módulo Cliente proposta no MobSec integra, em sua organização, componentes com funções de controle de acesso, gerenciamento de chaves e criptografia que em conjunto fornecem a confidencialidade e integridade dos dados.

O comportamento do Módulo Cliente para realizar esta missão, traduz-se em suas principais funções:

- Autenticação;
- Tratamento das requisições de acesso aos arquivos;
- Cifragem e decifragem dos dados.

A autenticação dos usuários para utilização do MobSec pode ser realizada de duas formas, conforme ilustra a Figura 16:

- Através do Módulo Cliente, quando não existir plataforma de serviços disponível no dispositivo móvel, ou

- Reutilizando os mecanismos oferecidos pela plataforma de serviços, onde a autenticação feita pela plataforma é repassada ao Módulo Cliente, tornando o processo de armazenamento seguro totalmente transparente para o usuário.

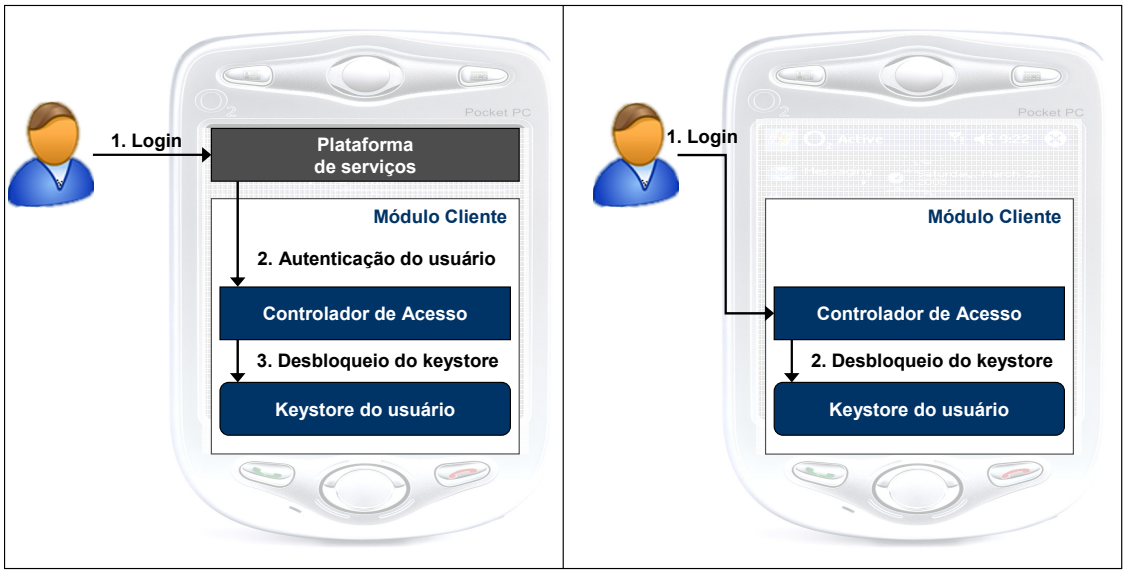

Figura 16: Autenticação do usuário.

O tratamento das requisições de acesso aos arquivos consiste na verificação da permissão de acesso aos dados, disponibilizando ou protegendo-os de usuários não autorizados.

As funções do Módulo Cliente são realizadas por sete componentes, apresentados na Figura 17, que integram a arquitetura dos dispositivos móveis:

- Secure Random Access File
- Controlador de Acesso
- CryptoManager
- HeaderManager
- *Keystores* dos usuários
- Arquivos cryptografados
- FileHeader de arquivos criptografados

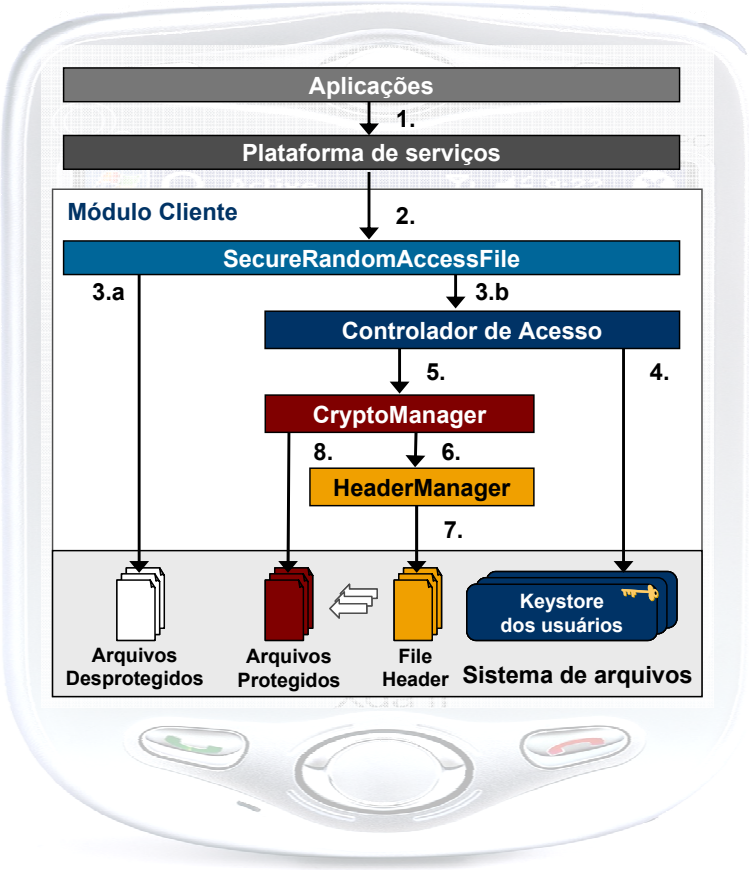

Figura 17: Arquitetura do Módulo Cliente no MobSec.

4.2.2.1. SecureRandomAccessFile

O Secure Random Access File (SRAF) tem por função receber todas as requisições para o acesso aos arquivos. O componente cria uma camada de abstração para o acesso aos dados no sistema de arquivos.

Esta abordagem permite que aplicações acessem os arquivos de maneira transparente, pois o SRAF se encarrega de verificar o tipo de arquivo que está sendo acessado, encaminhando a requisição de acesso diretamente ao sistema de arquivos ou ao Controlador de Acesso.

4.2.2.2. Controlador de Acesso

O Controlador de Acesso (CA) integra a lógica de segurança do sistema, atuando entre o SecureRandomAccessFile e o sistema de arquivo no recebimento dos chamados para leitura, criação e modificação dos dados protegidos.

Estes dados somente podem ser acessados se o usuário houver se autenticado com o CA, pois é durante o processo de autenticação que o *keystore* do usuário é desbloqueado, permitindo o uso das chaves armazenadas necessárias para a decodificação dos dados.

Por interceptar as requisições de acesso aos arquivos, o CA é capaz de verificar se o usuário possui direito de acesso, permitindo ou negando sua utilização. Se um usuário não autorizado tentar acessar o arquivo diretamente através do sistema de arquivo, evitando, portanto, os mecanismos do CA, este não será capaz de interpretar os dados, pois não terá acesso à chave necessária para decifrá-los.

4.2.2.3. Crypto Manager

O *Crypto Manager* (CM) é responsável por gerenciar todas as operações criptográficas da arquitetura do MobSec, sendo elas:

- cifragem e decifragem dos dados;
- Geração dos dados para bloqueio e desbloqueio do *keystore*.

Para executar as operações criptográficas, o *Crypto Manager* necessita de parâmetros, dentre eles os de definição do algoritmo criptográfico empregado e de tamanho da chave. Uma descrição detalhada dos parâmetros será apresentada no item 4.2.2.5.

Quando é feito um pedido de decodificação de um arquivo protegido, o CM primeiramente verifica os parâmetros utilizados para criptografar o arquivo, através de uma requisição de leitura do *File Header* feita ao *Header Manager*, realizando em seguida as operações criptográficas necessárias para recuperar os dados.

O processo de codificação dos dados ocorre de forma similar. O *Crypto Manager* recebe todos os parâmetros na cifragem do arquivo, através da requisição do Controlador de Acesso ou através de um arquivo de configurações, e finalmente cifra o arquivo. Em seguida requisita ao *Header Manager* que seja criado um *File Header*.

Além de codificação e decodificação dos dados, o CM também é responsável por gerar os dados para bloquear e desbloquear o *Keystore* do usuário, bem como gerar as chaves que cifram os arquivos protegidos, baseado nas chaves mestres armazenadas no *Keystore* do usuário.

4.2.2.4. Header Manager

O *Header Manager* é responsável pela criação e manipulação dos arquivos *File Headers*. Durante a cifragem de um determinado arquivo protegido o *Header Manager* recebe todos os dados referentes aos parâmetros usados e então cria um arquivo *XML Encryption* [16] contendo estas informações.

No processo de decodificação, o *Header Manager* lê o arquivo *File Header* referente ao arquivo que está sendo decodificado e extrai as informações necessárias, retornando-as ao *Crypto Manager*.

4.2.2.5. *File Header*

O File Header é uma arquivo XML [45] que segue a especificação do XML *Encryption* [16]. Ele é utilizado para descrever toda a informação de como um determinado arquivo foi cifrado, como por exemplo, a chave que foi empregada e o algoritmo. Cada arquivo cifrado possui um *File Header* correspondente. A Figura 18 ilustra a estrutura deste componente da arquitetura.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod Algorithm="AES/CBC/PKCS5Padding">
        <KeySize>192</KeySize>
        <ds:DigestMethod Algorithm='SHA' xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    </EncryptionMethod>
    <ds:KeyInfo>
        <ds:KeyName>MOREIRASAD</KeyName>
        <ss:KeyVersion xmlns:ss='http://www.sap.corp/projects/mobsec#'>
        1.0
        </KeyVersion>
    </KeyInfo>
    <CipherData>
        <CipherReference URI="file:///C:/exemplo.dat"/>
    </CipherData>
</EncryptedData>
```

Figura 18: Exemplo de *File Header*.

A especificação XML *Encryption* apenas define um número limitado de algoritmos criptográficos e apenas uma forma de "*padding*". Como um dos objetivos desta arquitetura é ser independente de algoritmos específicos, utiliza-se uma URI [42] no atributo

"Algorithm" do elemento *EncryptionMethod* que permita definir uma gama mais abrangente de algoritmos.

A especificação também define que os elementos filhos do elemento *EncryptionMethod* são determinados pelo valor da URI do atributo "*Algorithm*". Esta característica é melhor ilustrada na Figura 19 onde a definição do *schema* do elemento EncryptionMethod é apresentada.

```
<complexType name='EncryptionMethodType' mixed='true'>
    <sequence>
        <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
        <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
        <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
    </sequence>
    <attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

Figura 19: Definição de esquema (*schema*) do tipo EncryptionMethodType.

Este *schema* faz uso da construção ANY para permitir que tipos de outros "*name spaces*" sejam empregados como elementos filhos do tipo *EncryptionMethodType*. A garantia da integridade dos arquivos protegidos é feita através de um mecanismo baseado no "*hash"* do arquivo ou de partes do arquivo que será descrito com mais detalhe no ítem 5.1.4. A fim de especificar o algoritmo "*hash"* empregado faz-se uso da liberdade dada pela construção ANY acrescentando-se um elemento filho ao *EncryptionMethod*. No exemplo apresentado, o tipo do elemento filho foi re-aproveitado da especificação *XML Signature* [10] que define o componente *DigestMethod*.

Assim como a definição do *schema* do *EncryptionMethodType*, a definição do *KeyInfoType* também permite que elementos definidos em outros "*name spaces*" sejam empregados como elementos filho, conforme ilustrado na Figura 20. Esta característica pode ser utilizada para especificar a versão da chave empregada. No exemplo apresentado na Figura 18, o elemento "*KeyVersion*" foi utilizado para este fim.

```
<element name=”KeyInfo” type=”ds:KeyInfoType”/>
<complexType name=”KeyInfoType” mixed=”true”>
    <choice maxOccurs=”unbounded”>
        <element ref=”ds:KeyName”/>
        <element ref=”ds:KeyValue”/>
        <element ref=”ds:RetrievalMethod”/>
        <element ref=”ds:X509Data”/>
        <element ref=”ds:PGPData”/>
        <element ref=”ds:SPKIData”/>
        <element ref=”ds:MgmtData”/>
        <any processContents=”lax” namespace=”##other”/>
    </choice>
<attribute name=”Id” type=”ID” use=”optional”/>
</complexType>
```

Figura 20: Definição de esquema (*schema*) do tipo *KeyInfoType*.

A Figura 21 apresenta a definição do *schema* e do DTD do elemento *KeyVersion*.

```
Definição do Schema:
<element name=”KeyVersion” type=”string”/>


DTD:
<!ELEMENT KeyVersion (#PCDATA)>
```

Figura 21: Definição do esquema (*schema*) do elemento *KeyVersion*

4.2.2.6. Keystore

O *keystore* do usuário tem a função de armazenar todas as chaves mestres necessárias para executar as operações de codificação e decodificação dos arquivos protegidos. Cada usuário possui seu próprio *keystore* onde se encontram a chave mestre do usuário e as chaves mestres dos grupos aos quais o usuário pertence. Uma descrição mais detalhada da funcionalidade do *Keystore* pode ser encontrada em 4.2.1.1.

4.2.2.7. Tipos de Arquivo

A arquitetura do MobSec apresenta três tipos de arquivos para armazenamento de dados:

- Arquivos Desprotegidos;

- Arquivos do Usuário;
- Arquivos de Grupo.

Ao contrário dos demais tipos de arquivo; os Desprotegidos são armazenados sem a utilização de criptografia, não existindo, portanto, garantia de confidencialidade, integridade.

Os arquivos de Usuário e de Grupo são sempre protegidos, apresentando como fator diferencial, os direitos de acesso a eles associados:

- Arquivos de Usuários – são visíveis somente para o proprietário, sendo negado seu acesso a qualquer outro usuário que o solicite;
- Arquivos de Grupo – são compartilhados entre um grupo de usuários previamente definido, como por exemplo, um grupo de vendedores. Qualquer usuário não pertencente ao grupo terá negado o acesso ao arquivo. As permissões de acesso relacionadas ao arquivo permitem que um membro do grupo adicione novos usuários ao seu grupo.

A arquitetura proposta para o MobSec, considera que, para efetuar o processamento, o servidor sempre poderá recuperar os dados de todos os tipos de arquivos.

4.2.2.8. Comportamento do módulo cliente

O comportamento do módulo cliente está exemplificado na Figura 17, conforme descrição a seguir.

Quando uma aplicação solicita o acesso a um arquivo para a plataforma de serviços, conforme ilustrado pela seta 1, esta repassa a requisição para o SRAF, indicado pela seta 2. O SRAF recebe a requisição e se o arquivo solicitado é protegido, a encaminha para o Controlador de Acesso (CA), conforme indicado pela seta 3.b; para arquivos desprotegidos realiza um mapeamento para o sistema de arquivos, como indica a seta 3.a.

Na seqüência do processo, o CA verifica se o atual usuário possui permissão para acessar o arquivo, selecionando a chave adequada no *Keystore* do usuário, demonstrado pela seta 4. Uma vez que o usuário tenha a permissão de acesso, a requisição juntamente com a chave selecionada é encaminhada ao CryptoManager, que executa as funções criptográficas necessárias para o armazenamento ou recuperação dos dados, conforme indicado pela seta 5.

No processo de codificação dos dados, estes parâmetros são extraídos de um arquivo de configuração e passados ao CryptoManager. Na decodificação dos dados, a recuperação desta informação é feita através do Header Manager que acessa o File Header referente ao arquivo requisitado e extrai os parâmetros necessários, conforme esquematizado pelas setas 6 e 7.

Finalmente, o CryptoManager realiza as operações criptográficas necessárias para leitura ou escrita do arquivo protegido indicado pela seta 8, concluindo assim os procedimentos que contribuem para o armazenamento seguro.

4.2.3. Módulo Servidor

O Módulo Servidor implementado em uma máquina remota tem por objetivo disponibilizar recursos que permitam receber os dados enviados durante a sincronização e interpretá-los para processamento e envio ao *Backend* da Empresa. Para tanto, diversos componentes são necessários, entre eles o SGCS descrito no item 4.2.1.2. A definição da estrutura completa do Módulo Servidor foge ao escopo deste trabalho, que contempla e especifica apenas o SGCS.

4.3. Considerações em Relação à Literatura

A arquitetura do MobSec contempla aspectos relacionados ao gerenciamento de chaves, criptografia e recuperação de dados, que foram propostos a partir da análise da literatura clássica e atual pertinentes ao estado da arte dos sistemas de armazenamento seguro.

A geração das chaves dos usuários no servidor, além de impor maior segurança ao ambiente, acrescenta maior funcionalidade, quando comparado a sistemas como o CFS e o CryptoEx, pois permite a recuperação destas impedindo que dados contidos nos dispositivos sejam perdidos.

Outro aspecto relevante da arquitetura é que ao contrário de sistemas como o CryptoCache, não existe a necessidade do uso de hardware adicional, como *smartcards* nos dispositivos móveis, para a autenticação do usuário e armazenamento das chaves.

Alguns sistemas como o CFS e o AxCrypt impõem a necessidade de memorização de diversas senhas, muitas vezes excessivamente longas. Este tipo de abordagem adotado por estes sistemas não é conveniente para o usuário e por esta razão a arquitetura do MobSec

optou por um enfoque que busca reutilizar mecanismos já existentes permitindo que o serviço de armazenamento seguro seja totalmente transparente para o usuário e aplicações preservando ainda assim, a segurança dos arquivos.

4.4. Conclusões do Capítulo

Neste capítulo foi apresentada a arquitetura do MobSec, abordando os componentes integrantes e suas principais características, finalidade e mecanismos. A arquitetura proposta é composta por dois módulos, Módulo Servidor e Módulo Cliente, que residem respectivamente, no servidor e nos dispositivos móveis.

O MobSec apresenta recursos que permitem o armazenamento seguro de dados durante um ciclo completo do processo que inicia com a geração das chaves do usuário no servidor e encerra após a conclusão do processo de sincronização dos dados protegidos com o servidor.

Capítulo 5: Aspectos da Implementação

Para se verificar a eficácia e a confiabilidade da arquitetura do MobSec, uma implementação foi desenvolvida para ser agregada ao *framework Mobile Infrastructure (MI)*. Este *framework* é um produto desenvolvido pela empresa SAP para dispositivos móveis tais quais *smart phones*, PDAs e *laptops*. Este produto oferece um conjunto de serviços para desenvolvimento de aplicativos voltados para dispositivos móveis, como por exemplo, o serviço de persistência de dados.

Toda esta infra-estrutura faz parte de um sistema que estende as funcionalidades da empresa para os usuários, estando eles conectados à rede ou não. Esta infra-estrutura também fornece um serviço para a sincronização dos dados do dispositivo móvel com o servidor. Durante a sincronização os dados são transmitidos utilizando o protocolo SSL [49], garantido assim a segurança dos dados na transmissão. Porém, enquanto os dados não são sincronizados eles permanecem em texto em claro no dispositivo.

O MobSec foi adaptado ao *Mobile Infrastructure* de forma a fornecer o serviço de armazenamento seguro, de maneira transparente para o usuário.

Este capítulo tem por objetivo descrever a implementação e integração da arquitetura proposta, apresentar as ferramentas escolhidas, verificar o desempenho, bem como avaliar a viabilidade da utilização do modelo implementado na indústria.

5.1. Protótipo Implementado

O protótipo implementado é composto por dispositivos móveis, um servidor de sincronização dos dados e o *Backend* da empresa conforme ilustra a Figura 22.

Figura 22: Protótipo implementado.

As aplicações instaladas nos dispositivos móveis encontram-se arquiteturalmente acima do *framework* do *Mobile Infrastructure*, conforme apresentado na Figura 23. Estas aplicações utilizam as facilidades oferecidas pelas interfaces públicas deste *framework* que oferecem uma série de serviços, entre eles o serviço de sincronização e o serviço de persistência.

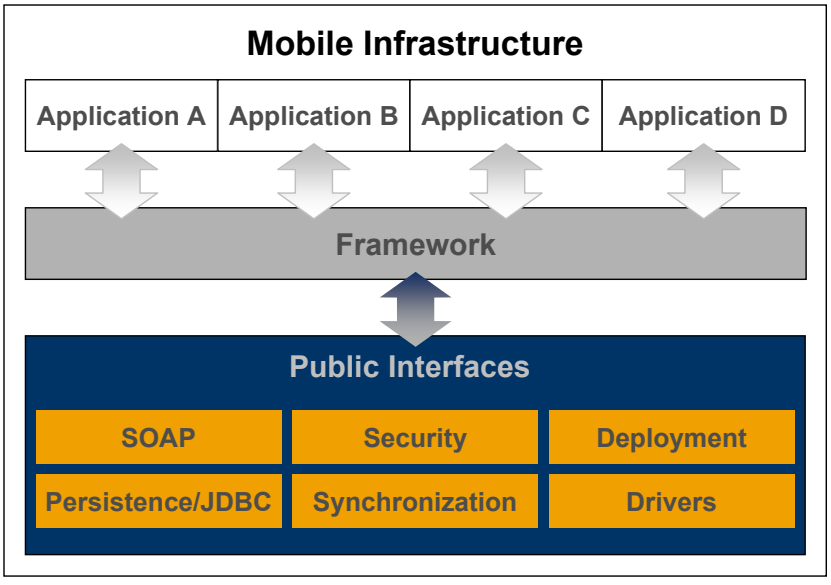

Figura 23: Framework do *Mobile Infrastructure*.

O serviço de persistência permite que dados sejam armazenados no dispositivo móvel para posterior sincronização com o servidor. O protótipo implementado visa verificar a usabilidade da integração do MobSec com o serviço de persistência do *Móbile Infrastructure*.

O exemplo adotado para o protótipo é uma aplicação de catálogo de endereços que é disponibilizada para o usuário através do *Mobile Infrastructure*. Esta aplicação permite que os contatos sejam disponibilizados somente para o usuário ou para um grupo de usuários.

O MobSec foi todo implementado em Java (JDK1.2.2), com o intuito de ser compatível com o *Personal Java*. Utilizou-se a ferramenta Eclipse para auxiliar o desenvolvimento desta aplicação.

As operações de *parse* e escrita dos arquivos XML são feitos pela biblioteca Crimson. Esta biblioteca é um projeto ApacheXML [1] já utilizada pela infra-estrutura do *Mobile Infrastructure.* Uma implementação simplificada de um *parser* XML *Encryption* foi elaborada de forma a satisfazer os requisitos para escrita e leitura dos arquivos *File Header*.

5.1.1. Integração com o *Mobile Infrastructure*

Buscando fornecer o serviço de armazenamento seguro ao *Mobile Infrastructure* integrou-se o MobSec ao serviço de persistência de dados conforme apresentado na Figura 24.

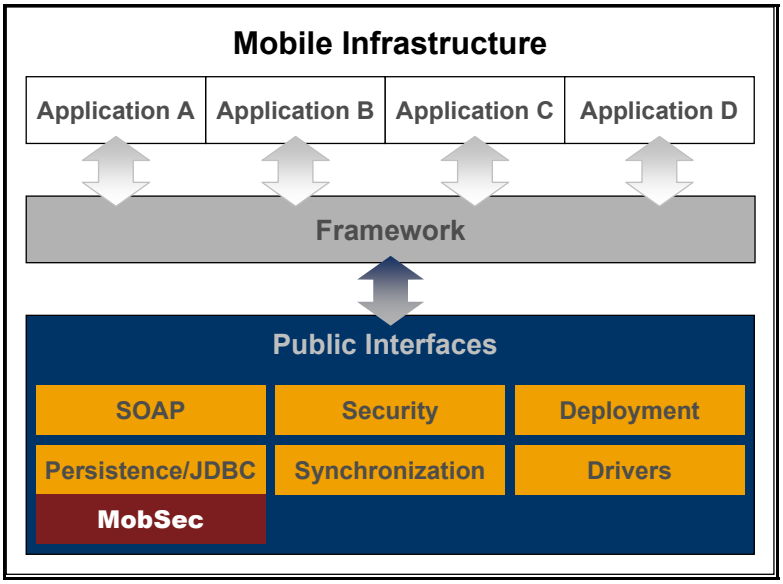

Figura 24: Integração do MobSec ao *Mobile Infrastructure*

Uma segunda adaptação foi realizada no sistema de autenticação do *Mobile Infrastructure*. Para um usuário utilizar esta infra-estrutura ele deve primeiramente se conectar ao sistema se autenticando através de um *login* e senha. Após a autenticação ele recebe direito de acesso a todas as aplicações instaladas. No sentido de manter a utilização desta infra-estrutura conveniente para o usuário, optou-se por reutilizar esta senha para gerar as informações necessárias para o desbloqueio do *keystore* do usuário.

Após a autenticação, o usuário tem acesso a todas as aplicações instaladas, incluindo as que utilizam o serviço de armazenamento seguro. Ao terminar de utilizar as aplicações o usuário se desconecta para evitar que as aplicações fiquem disponíveis. Existe também um mecanismo que desconecta o usuário caso esteja inativo por um determinado período de tempo. Este processo de desconexão, seja ele automático ou por intermédio do usuário também é informado ao MobSec que remove todas as referências existentes relacionadas ao *keystore* do usuário.

A geração da chave que codifica e decodifica o *keystore* dos usuários é feita utilizando as informações do *login* e senha do usuário como parâmetros de entrada para uma função de derivação de chave definida no RFC2246. Esta função de derivação de chave foi escolhida, pois emprega dois algoritmos de *hash* o que garante a segurança da senha do usuário mesmo que um dos algoritmos seja comprometido.

5.1.2. Imposição do Controle de Acesso

O uso de ACLs ou Listas de competência são mecanismos de controle de acesso adequados a sistemas distribuídos, entretanto o seu emprego em uma máquina local (dispositivo móvel) somente tem sentido se for possível impor as políticas de controle de acesso.

Quando se emprega criptografia simétrica aos dados, a partir do momento em que a chave se torna acessível para o usuário os arquivos podem ser lidos ou escritos. Por esta razão o uso de ACLs ou Listas de competência como descritas em 2.2.4 não é adequado, pois as políticas de segurança não poderão ser impostas. Sendo assim, o MobSec não faz diferenciação entre leitores e escritores, sendo o mecanismo de controle de acesso implementado baseado nas chaves disponíveis no *keystore* do usuário.

Ao tentar acessar um arquivo, o Controlador de Acesso verifica qual a chave necessária para efetuar a operação. Se a chave necessária estiver disponível, o acesso é permitido, caso contrário o acesso é negado.

Quando o usuário se desconecta do *Mobile Infrastructure*, por vontade própria ou por inatividade, o Controlador de Acesso remove todas as informações referentes ao *keystore* e a autenticação do usuário para evitar acesso não autorizado aos arquivos.

5.1.3. Características do *Crypto Manager*

O *Crypto Manager* realiza todas as operações criptográficas da arquitetura do MobSec conforme descrito em 4.2.2.3. Para efetuar estas operações utilizou-se a infra-estrutura de *Security Provider* definido pelo Java da Sun. O pacote JCE-IAIK [13] foi escolhido para fornecer a implementação dos algoritmos criptográficos. Este pacote foi desenvolvido no *Institute for Applied Information Processing and Communications* da Universidade Tecnológica de Graz, na Áustria. A escolha por este *Security Provider* deu-se devido à experiência prévia da empresa SAP com o mesmo.

Além de fornecer a implementação dos algoritmos criptográficos, este pacote também fornece uma implementação de *keystore* que permite utilizar uma chave para cifrá-lo. Por fornecer todas as funcionalidades necessárias, esta implementação de *keystore* foi adotada.

5.1.4. Formato dos arquivos protegidos

Foram implementados dois formatos de arquivo para a cifragem dos dados que serão posteriormente comparados: cifragem simples e cifragem em blocos.

Na cifragem simples, para garantir a integridade dos arquivos, primeiramente calcula-se o *hash* dos dados do texto em claro e agrega-se o valor calculado ao início do texto em claro conforme ilustrado na Figura 25.

A confidencialidade é fornecida cifrando o *hash* juntamente com o texto em claro. A chave usada nesta etapa depende do tipo do arquivo que está sendo gerado. Quando os dados são de um arquivo de usuário, a chave usada é a chave do usuário. Da mesma forma, quando o arquivo é de grupo, a chave usada deve ser a chave do grupo que compartilhar este arquivo.

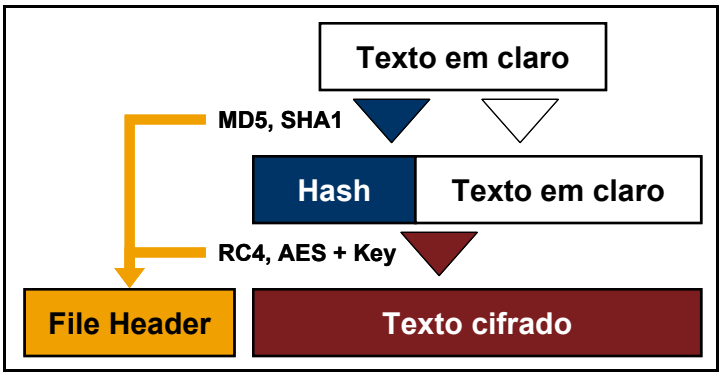

Figura 25: Proteção dos dados.

Após selecionar a chave, o arquivo é cifrado e armazenado no sistema de arquivos, juntamente com um arquivo *File Header* que contém os parâmetros necessários para se decodificar o arquivo posteriormente.

A Figura 26 apresenta o formato de um arquivo protegido por cifragem de blocos e os passos necessários para sua criação. Neste formato, os dados são separados em blocos, e então codificados. Esta abordagem permite que o impacto do processo de cifragem dos dados seja reduzido em situações onde somente uma parte do arquivo é lida ou escrita. Um outro aspecto é que a memória necessária neste processo é reduzida, permitindo a manipulação de arquivos maiores.

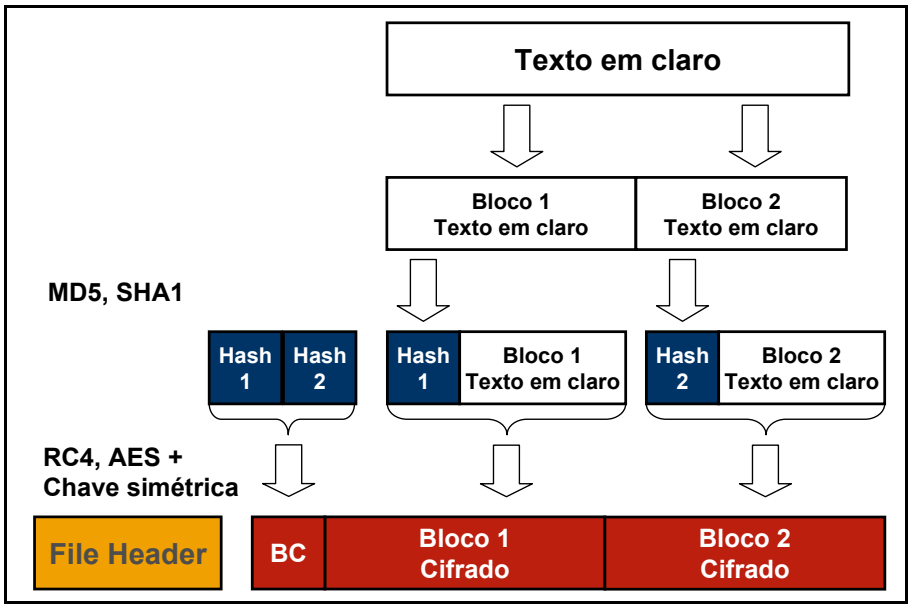

Figura 26: Formato dos arquivos protegidos

Assim como a cifragem simples, o mecanismo utilizado para cifrar os arquivos de usuário e de grupo é o mesmo, variando apenas a chave simétrica utilizada (chave do grupo, ou chave do usuário).

O *hash* de cada bloco de texto em claro é calculado para garantir a integridade do bloco. Em seguida o *hash* é concatenado com o bloco de dados e criptografado com uma cifra simétrica gerando blocos cifrados. Os blocos cifrados são concatenados e armazenados no sistema de arquivo juntamente com um arquivo *File Header* que especifica os parâmetros usados durante o processo de codificação dos dados.

Este procedimento apenas garante a integridade dos blocos de dados o que não impede que blocos cifrados sejam intercambiados, corrompendo assim a integridade do arquivo como um todo.

Para detectar ações deste tipo, o valor do *hash* de cada bloco *plaintext* é concatenado na ordem dos blocos e então cifrado com a mesma chave que codifica o resto dos dados. O resultado desta operação é chamado de *"Block Control"* e é adicionado ao início dos dados cifrados, como pode ser visualizado na Figura 26. Quando um bloco é decifrado, o *hash* contido dentro do bloco cifrado é comparado com aquele contido dentro do *Block Control* para verificar a integridade do arquivo.

5.2. Resultados obtidos

O módulo cliente foi implementado por completo tendo sido integrado ao *Mobile Infrastructure*. A aplicação de catálogo de endereços foi corretamente instalada via sincronização com o servidor e os dados recebidos foram cifrados para persistência pelo serviço de armazenamento seguro do MobSec.

Em um primeiro momento, foi testada a imposição do controle de acesso que ocorreu da forma esperada, permitindo ou negando o acesso do usuário de acordo com as permissões de acesso deste.

Em seguida testou-se a verificação da integridade dos dados alterando os bytes do arquivo cifrado diretamente no sistema de arquivo. Como resultado a aplicação acusou que o arquivo estava corrompido.

Finalmente uma série de testes de desempenho foram realizados e serão discutidos no item a seguir.

5.2.1. Testes de Desempenho

Para avaliar o desempenho da implementação do MobSec realizou-se testes de desempenho para leitura e escrita de arquivos com os algoritmos descritos na Tabela 4.

Tabela 4: Algoritmos empregados nos testes de desempenho

Algoritmo	Modo	*Padding*	Tamanho da chave
3DES	CBC	PKCS5Padding	128
Blowfish	CBC	PKCS5Padding	128
AES	CBC	PKCS5Padding	128

Estes testes foram feitos em um iPaq série H3900, com as seguintes especificações:

- Processador 400 MHz Intel PXA250
- 64 MB RAM

Cada valor aqui apresentado é o resultado da média de 5 medidas. A máquina virtual Java empregada para os testes foi o Jeode, que foi especialmente projetada para *PocketPCs*.

Fez-se uma comparação entre a cifragem simples de arquivo e a cifragem de blocos de arquivo com tamanhos de bloco variando de 1KB a 1MB. Cabe ressaltar que na cifragem simples somente foi possível manipular arquivos de até 1MB devido a restrições de memória existente neste aparelho, já na cifragem de blocos trabalhou-se com arquivos de 10MB.

Para contornar estas diferenças adicionou-se uma linha de tendência aos valores encontrados com a cifragem simples que permitem projetar qual seria um provável desempenho deste método caso houvesse memória suficiente no dispositivo para manipular arquivos maiores.

Os gráficos da Figura 27 apresentam o tempo gasto para a leitura dos bytes de um arquivo cifrado empregando o algoritmo *Triple DES* como cifra simétrica e o SHA-1 como algoritmo *hash*. O primeiro gráfico apresenta uma visão do desempenho da leitura dos dados até 10MB enquanto que o segundo se concentra em apresentar os dados colhidos somente até 2MB para que os detalhes da inicialização da leitura possam ser analisados.

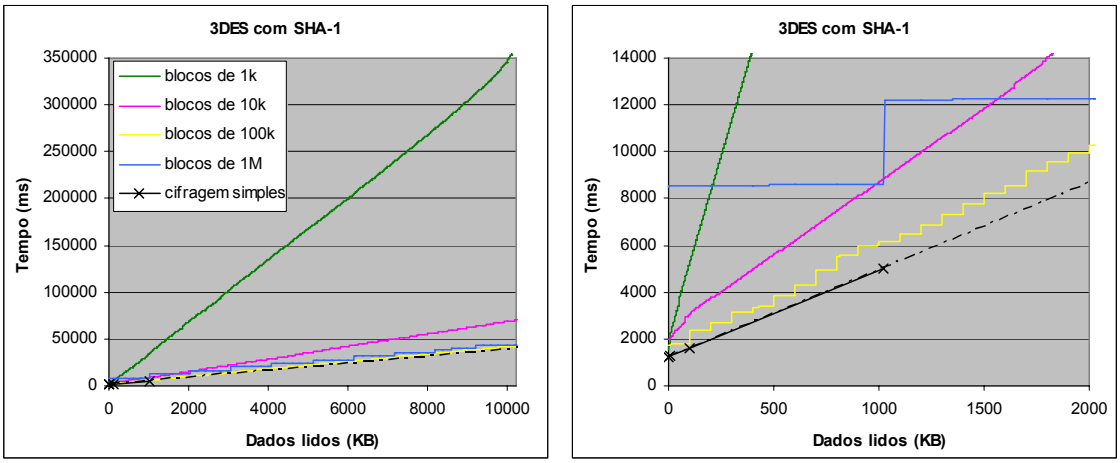

Figura 27: Leitura de um arquivo empregando 3DES com SHA-1.

Observa-se que a cifragem simples é linear, enquanto que a cifragem de bloco apresenta degraus. Este fenômeno ocorre, pois o MobSec decifra o bloco de dados e armazena o resultado na memória, fornecendo os bytes a medida que são solicitados. Quando um byte de um outro bloco é requisitado, o novo bloco é decifrado e armazenado na memória causando os degraus. Isto explica a diferença entre o tamanho dos degraus, pois de acordo com o tamanho dos blocos, estes são carregados com maior ou menor freqüência.

Nestes gráficos também é possível verificar que o tempo necessário para inicializar a leitura dos dados pouco varia entre os blocos de 1K e 100K, sendo o valor em torno de 2 segundos. Entretanto para inicializar a leitura de um arquivo empregando blocos de 1MB, gasta-se aproximadamente 8 segundos.

Por outro lado, blocos de 100K têm um desempenho semelhante aos blocos de 1MB para leitura de maiores volumes de dados e não apresentam este *overhead* inicial. 10MB de dados são lidos em 42,5 segundos pela configuração de blocos de 100KB enquanto que com blocos de 1MB são necessários 43 segundos.

De acordo com a análise feita a partir do gráfico da Figura 27, verifica-se que para leitura de maiores volumes de dados, a cifragem simples é a que apresenta melhor desempenho. Os blocos de 100KB possuem um desempenho semelhante, sendo necessários 10 segundos para a leitura de 2MB de dados cifrados, enquanto que na cifragem simples seriam necessários aproximadamente 9 segundos.

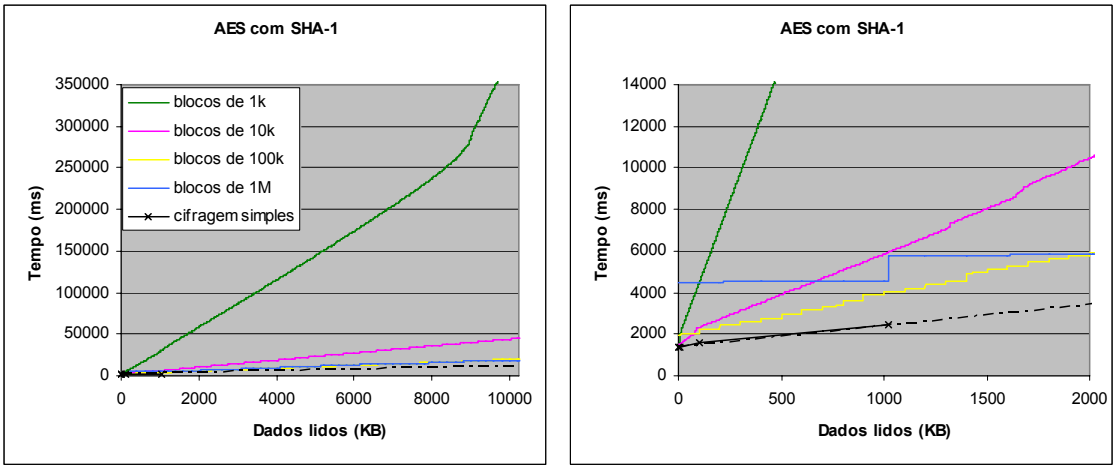

Figura 28: Leitura de um arquivo de 10MB empregando AES com SHA-1.

A Figura 28 demonstra os resultados obtidos na leitura de dados cifrados utilizando a infra-estrutura do MobSec. Para a realização deste teste foi empregado o algoritmo *AES* como cifra simétrica e o SHA-1 como algoritmo *hash*. No primeiro gráfico é demonstrado o resultado da leitura dos dados até 10MB. O segundo gráfico apresenta os resultados do desempenho da leitura dos dados até 2MB o que permite analisar com mais clareza o que ocorre na inicialização do processo.

É possível verificar que ocorrem os mesmos fenômenos apresentados na Figura 27, a leitura dos dados, quando empregada a cifragem de blocos apresenta degraus devido ao carregamento de decifragem dos mesmos.

A utilização de blocos de 1K resulta numa perda de desempenho considerável. Isto ocorre devido ao intenso cálculo de *hashs* necessários por conseqüência do pequeno tamanho do bloco.

A análise do gráfico também permite verificar que a inicialização da leitura dos dados para a configuração de blocos de 1MB, leva aproximadamente 4,5 segundos, enquanto que os blocos de 1KB a 100KB levam aproximadamente 2 segundos.

Pode-se concluir a partir dos gráficos apresentados, que dado um volume de dados fixo, o tempo gasto para a decifragem dos dados diminui à medida que aumenta o tamanho do bloco empregado. Por outro lado, o tempo de inicialização aumenta à medida que cresce o tamanho do bloco usado.

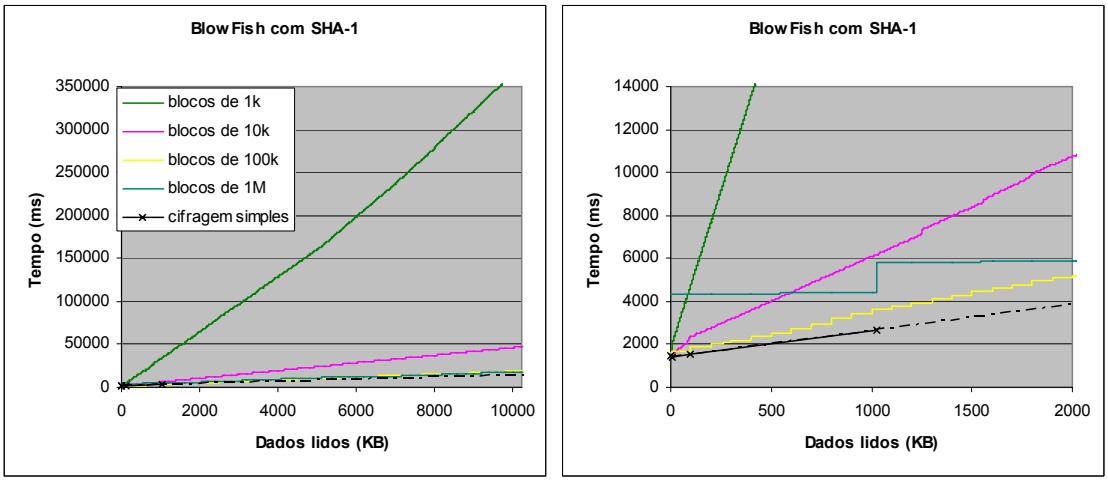

Figura 29: Leitura de um arquivo de 10MB empregando BlowFish com SHA-1.

Finalmente, a Figura 29 apresenta os gráfico obtidos a partir dos resultados do teste de leitura dos arquivos protegidos pela infra-estrutura do MobSec, utilizando o algoritmo *Blowfish* como cifra simétrica, e o algoritmo SHA-1 como *hash*.

Neste algoritmo a inicialização dos blocos de 1MB é feita em aproximadamente 4,5 segundos, enquanto que as demais configurações de blocos levam em torno de 1,8 segundos.

A partir do gráfico é possível concluir que a cifragem simples sempre possui um desempenho superior se comparada as cifragens de bloco. Entretanto, se levarmos em consideração que a cifragem de blocos permite decifrar somente um bloco independentemente da posição onde este se encontra, irão existir situações onde esta terá uma resposta mais rápida às requisições de leitura, pois não será necessário decifrar todo o arquivo até a posição onde os dados se encontram como é o caso da cifragem simples.

O gráfico da Figura 30 apresenta uma comparação entre os algoritmos *AES, Triple DES e Blowfish* para blocos de 1KB. Conclui-se, a partir dos resultados obtidos, que não existe variação significativa entre o desempenho dos algoritmos. Isto ocorre, pois para um bloco pequeno como o aqui analisado, a diferença do desempenho devido ao algoritmo simétrico é ofuscado pelo tempo do calculo do *hash* de cada bloco.

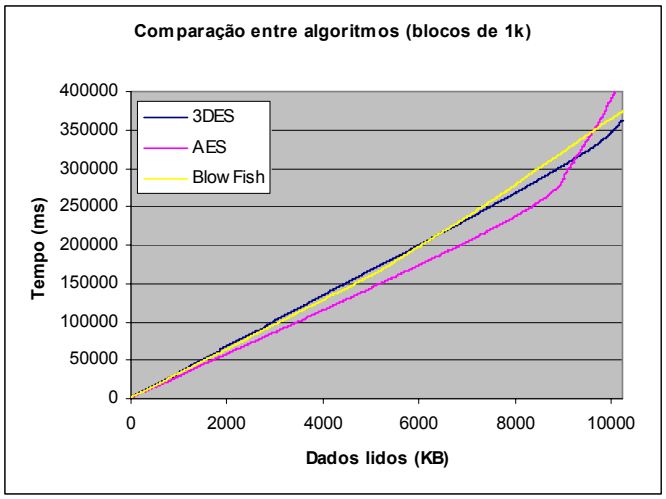

Figura 30: Comparação entre algoritmos para blocos de 1KB

Nos gráficos apresentados na Figura 31, onde é feito uma comparação entre os três algoritmos para blocos de 10KB e 100KB, a diferença do desempenho dos algoritmos se torna mais evidente. Nestes gráficos o *Triple DES* se diferencia dos demais por possuir um desempenho muito inferior, levando 70 segundos usando blocos de 10KB para decifrar um arquivo de 10MB, enquanto que o *AES* e o *Blowfish* levam 47,2 e 45,4 segundos respectivamente.

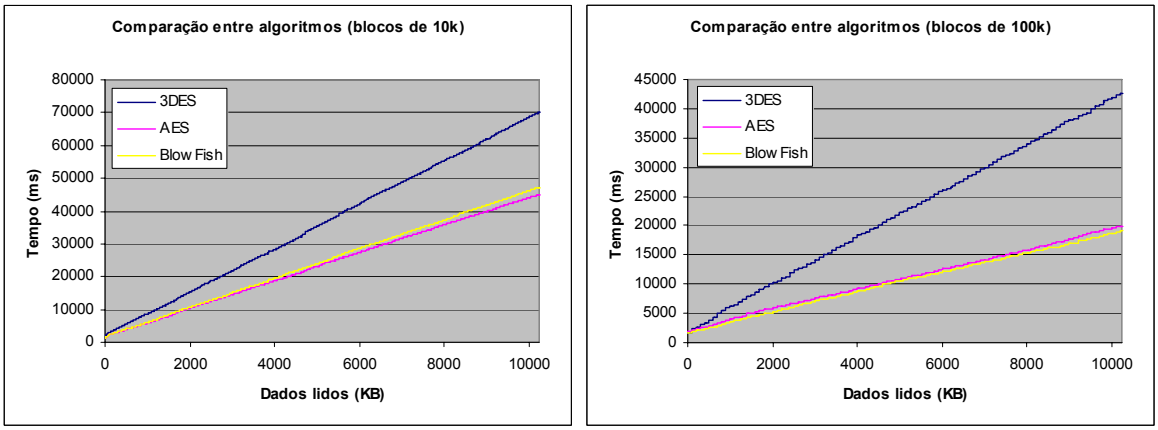

Figura 31: Comparação entre algoritmos para blocos de 10KB e 100KB.

No emprego de blocos de 100KB, o *Triple DES* levou 42 segundos para decifrar um arquivo de 10MB enquanto que o *AES* e o *Blowfish* gastaram 20 segundo aproximadamente. Pode-se também perceber a linearidade dos resultados obtidos, pois cada algoritmo necessita de um tempo médio padrão para carregar um novo bloco na memória.

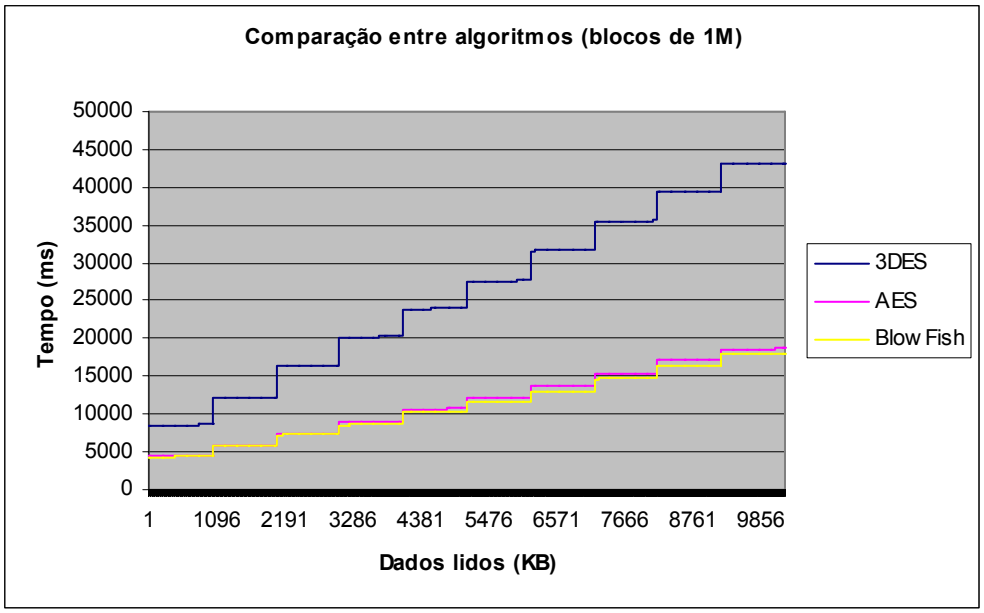

Figura 32: Comparação entre algoritmos para blocos de 10KB e 100KB.

A Figura 32 apresenta uma comparação entre os resultados obtidos na leitura de dados usando blocos de 1MB. Analisando este gráfico verifica-se que o *AES* tem um desempenho um pouco superior ao *Blowfish*. Esta diferença no desempenho não é muito significativa se comparada à melhoria existente em relação ao *Triple DES*, que leva 43 segundos para decifrar 10MB enquanto que o AES e o Blowfish levam em torno de 18 segundos.

5.2.2. Considerações

O serviço de gerenciamento de chaves do lado do servidor conforme descrito em 4.2.3 foi implementado por completo, porém não pôde ser integrado ao servidor de sincronização. Desta forma, no protótipo implementado, as chaves são geradas e adicionadas ao *keystore* do usuário por uma aplicação stand-alone. A transferência do *keystore* para o dispositivo é feita manualmente através de uma memória removível.

Devido à impossibilidade de integração, os dados protegidos enviados ao servidor precisam ser primeiramente decodificados no dispositivo móvel antes de serem enviados ao servidor para que este possa interpretar os dados recebidos.

Um outro aspecto da implementação que cabe ressaltar é que apesar da integridade do arquivo poder ser verificada isto não impede que o arquivo como um todo seja substituído ou apagado. O sistema de arquivo de PocketPcs (Windows CE) não possui definição de usuários, tão pouco de permissão de acesso a arquivos. Além disso, em

dispositivos como laptop, é possível acessar os dados armazenados através de outro sistema de arquivos como o do Linux burlando assim a proteção contra escrita imposta pelo sistema operacional. Por estas razoes não é possível evitar que este tipo de ataque ocorra.

5.3. Conclusões do Capítulo

A implementação do MobSec demonstrou ser de extrema importância realizar uma análise de desempenho com variados tamanhos de blocos, pois a partir desta informação é possível definir qual a melhor configuração a ser adotada para uma determinada aplicação, pois existe um compromisso entre desempenho, tamanho de blocos e volume de dados acessados. Deve-se, portanto, realizar um estudo sobre a origem dos dados acessados a fim de definir a configuração que será empregada.

O exemplo de aplicação se comportou conforme esperado, permitindo ou recusando o acesso aos dados dos conforme os direitos de acesso do usuário. A aplicação também percebeu alterações realizadas sem autorização nos arquivos informando que a integridade dos dados havia sido corrompida. Os requisitos de confidencialidade e integridade dos dados foram, portanto, cumpridos.

Capítulo 6: Considerações Finais e Perspectivas Futuras

Esta dissertação se constitui na proposta do MobSec, uma arquitetura para armazenamento seguro de dados em dispositivos móveis, onde se procurou contemplar a conjugação dos fatores usabilidade, desempenho, integridade e confidencialidade.

Nos Cap. 2 e 3 foram apresentados os fundamentos teóricos do armazenamento seguro de dados e o estado da arte destes sistemas, resultantes de ampla pesquisa em bibliografia emergente, reconhecida e aceita na comunidade científica. O estudo permitiu comparar as características dos atuais sistemas e constatar a inadequação destes quando aplicados aos dispositivos móveis, no que diz respeito as suas arquiteturas e premissas conforme apresentado nas considerações do Cap. 3.

O modelo MobSec apresentado nos Cap. 4 e 5 se constitui num sistema de armazenamento seguro de dados que visa um cenário *offline* onde o usuário insere informações sigilosas em um dispositivo móvel para posterior sincronização com um servidor.

Segundo Blaze este sistema é classificado como um modelo a nível de aplicação, mas com a vantagem de permitir o compartilhamento de arquivos entre diversas aplicações, pois funciona como um serviço de um *framework* que suporta diversas aplicações. Na classificação de Riedel, esta arquitetura é considerada conveniente, pois nenhuma senha extra é solicitada ao usuário.

A arquitetura proposta e os testes realizados permitem afirmar que o modelo MobSec é um suporte de armazenamento seguro que responde de forma eficiente à necessidades de manter a inviolabilidade de informações críticas. Esta arquitetura buscou atender às

restrições impostas aos dispositivos móveis, levando em consideração principalmente sua baixa capacidade de processamento.

Desta forma, no que diz respeito ao contexto computacional para dispositivos móveis, este sistema fornece confidencialidade e integridade dos dados buscando atender aos requisitos de usabilidade e desempenho do sistema MobSec.

No atual estágio de desenvolvimento dos sistemas de armazenamento seguro de dados, o projeto MobSec apresenta-se como inovador no que diz respeito ao seu gerenciamento de chaves.

Diversos testes foram realizados para comprovar a viabilidade da proposta, demonstrando que o projeto MobSec atendeu às expectativas desta pesquisa, constituindo-se hoje em uma patente registrada pela SAP e a autora deste projeto nos Estados Unidos (*Method and Apparatus for securely storing data on the application layer in Mobile Devices*).

Este trabalho tratou especialmente as ameaças de roubo e perda de dispositivos móveis e a proteção e recuperação de dados críticos neles armazenados. Esperamos que as soluções apresentadas tenham contribuído para melhorar a segurança destes dispositivos.

6.1. Perspectivas Futuras

Existe uma gama de possibilidades para o aprimoramento deste trabalho, a mais imediata seria a integração da funcionalidade desenvolvida para a aplicação *stand-alone* para a criação dos *keystore* no servidor.

Está sendo realizado um estudo para viabilizar a implementação da arquitetura do MobSec em um produto da SAP. Diversos testes já foram realizados e parte da arquitetura migrada.

Referências Bibliográficas

[1] Crimson.jar. Java XML Parser. The Apache Software Foundation. Julho 2004. http://xml.apache.org/crimson/

[2] AxCrypt.exe. 1.6.0. Programa para a cifragem de dados. Svante Seleborg. UFSC, EEL, LCMI. Florianópolis, Janeiro de 2005. http://axcrypt.sourceforge.net/,

[3] Bell, D.E. e La Padula, L. J. *Secure Computer Systems: Mathematical Foundations*. The MITRE Corporation, Bedford Massachusetts, MTR-2547, v. 1, Mar.1973.

[4] Blaze, M. A. Cryptographic File System for Unix. In: First ACM Conference on Computer and Communications Security (Nov. 1993: Fairfax ,VA). *Proceedings*. VA, Nov. 1993.

[5] Cattaneo, G.; Catuogno, L.; Del Sorbo, A. et al. The design and implementation of a Transparent Cryptographic File System for Unix. In: USENIX Technical Conference 2001. *FREENIX Track Technical Program*, 2001.

[6] CryptoEX. Programa para PocketPC com a finalidade de cifrar dados. Glück & Kanja Group. http://www.cryptoex.com/cryptoexclassic/pocket.aspx?lang=en

[7] Daemen, J.; Rijmen, V. *AES Proposal: Rijndael.* Disponível na Internet, último acesso em dezembro de 2004. http://csrc.nist.gov/CryptoToolkit/aes/round2/r2algs.htm

[8] DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, FIPS PUB 46-2 Data Encryption Standard (DES) U.S. Dezembro 1993

[9] Dierks, T.; Allen, C. *The TLS Protocol* (rfc 2246). IETF Request for Comments, Janeiro de 1999.

[10] Eastlake, D; Reagle, J. e Solo, D. *XML-Signature Syntax and Processing W3C Recommendation*, Fevereiro de 2002. Disponível na Internet, último acesso em Dezembro de 2004. http://www.w3.org/TR/xmldsig-core/

[11] Forman, G. H. e Zahorjan, J. The Challenges of Mobile Computing. *IEEE*, 1994.

[12] Goguen, J. A. e Mesajuer, J. *Security Policies and Security Models*. In: Proceedings of IEEE Symposium on Research in Security and Privacy, 1982.

[13] IAIK. Security Provider. Institute for Applied Information Processing and Communications. Maio de 2004. http://jce.iaik.tugraz.at/products/index.php

[14] Shirley, R. *Internet Security Glossary* (rfc 2828). IETF Request for Comments, Maio de 2000

[15] Imamura, T; Clark, A. e Maruyama, H. *A Stream-based Implementation of XML Encryption*. In: ACM, Workshop on XML Security. (2002: Washington, DC). Washington, DC: ACM Press, 2002.

[16] Imamura, T; Dillaway, B. e Simon, E. *XML Encryption Syntax and Processing*, Dezembro de 2002. Disponível na internet, último acesso em Setembro de 2004. http://www.w3.org/TR/xmlenc-core/

[17] Jansen, W A. Authenticating Users on Handheld Devices. In: Canadian Information Technology Security Symposium (Mai. 2003). *Proceedings*. Maio 2003

[18] Lampson, B. W. Protection. In: 5th Princeton Conference on Information Sciences and Systems. (1971: Princeton). *Proceedings*. Princeton, 1971. p. 437

[19] Lampson, B.; Abadi, M.; Burrows, M. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems* v.**10**, n.4 p. 265-310, Novembro 1992.

[20] Landwehr C. E. Computer Security. IJIS, v. 1, p. 3-13, Jul. 2001.

[21] Bishop, M.; *Computer Security: Art and Science*. California: AWP Publisher, 2002

[22] Mavridis, I; Pangalos, G. Security Issues in a Mobile Computing Paradigm. Informatics Laboratory, Computer Division, Faculty of Technology, University of Thessaloniki.

[23] Microsoft Corporation. *Encrypting File System EFS.* Disponível na Internet, último acesso em Setembro de 2004. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/encrypt_overview.mspx

[24] Callas, J.; Donnerhacke, L.; Finney, H. et al. OpenPGP Message Format (rfc 2440). IETF Request for Comments, Novembro 1998.

[25] Osborn, S.; Sandhu, R. e Munawer, Q. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security*, v. 3, n. 2, p. 85-106, Maio de. 2000.

[26] Otterloo, S. V. *A Security Analysis of Pretty Good Privacy*, Setembro de 2001. Disponível na Internet, último acesso em Janeiro de 2005. http://citeseer.ist.psu.edu/639221.html

[27] Pfitzmann, A.; et al. Mobile User Devices and Security and Security Modules: Design for Trustworthness. IBM Technical Repor RZ-2784

[28] Zimmermann, P. *Pretty Good Privacy*. Disponível na Internet, última visita em dezembro de 2004. http://www.philzimmermann.com/EN/background/index.html 1991

[29] PGPDISK. Programa para a cifragem de dados baseado em PGP. UFSC, EEL, LCMI. Florianópolis, Novembro de 2004. http://www.pgpi.org/products/pgpdisk/

[30] Riedel, E; Kallahalla, M. e Swaminathan, R. A framework for evaluating storage systems security. In: 1st Conference on File and Storage Technologies (FAST). (Monterey, CA: Jan. 2002). *Proceedings*. Monterey, CA, 2002

[31] RSA Laboratories. *PKCS #1 – RSA Cryptography Standard*. v. 21, Junho de 2002.

[32] RSA Laboratories. *PKCS #5 v2.0 Password-Based Cryptography Standard*. Março de 1999.

[33] Russel, D. e Gangemi, G. T. *Computer Security Basics*. 1ed. Sebastopol, CA: O'Reilly & Associates, 1992.

[34] Santin, A. O. *Teias de Federações: uma abordagem baseada em cadeias de segurança, para autenticação autorização e navegação de sistemas de larga escala*. Florianópolis, 2004. Tese de doutorado em Engenharia Elétrica - Centro Tecnológico, Universidade Federal de Santa Catarina,

[35] Sandhu, S. R. e Samarati, P. Access Control: Principles and Practice. *IEEE Comunications Magazine*, p. 40-48, Set. 1994.

[36] Sandhu, S. R. e Samarati, P. Authentication, Access Control, and Audit. *ACM Computing Surveys*, v. 28, n. 1, Março de 1996.

[37] Jensen, C.D. CryptoCache: A Secure Sharable File Cache for Roaming Users. In: European Workshop Beyond the PC new chalanges for operating systems. *Proceedings of the 9th Workshop on ACM SIGOP,* Nova York, 2000.

[38] Schneier, B Description of a New Variable-length Key, 64-bit Block-Cipher (Blowfish). In: Fast Software Encryption, Cambridge Security Workshop (December 1993). *Proceedings*, Springer-Verlag, 1994, p. 191-204.

[39] Schneier, Bruce. *Applied Cryptography: Protocols, algorithms and source code in C*. 2 ed, New York: John Wiley & Sons, 1996.

[40] Shamir, Adi. How to Share a Secret. *Communications of the ACM*, v. 22, n. 11, p.612-613, Nov. 1979

[41] Steiner, J.G; Neuman, C. E Schiller, J.I. Kerberos: Autentication Service for Open Network Systems. In: Winter USENIX. *Proceedings*, 1988.

[42] Berners-Lee, T.; Fielding, R.; Masinter, L. Uniform Resource Identifiers: Generic Syntax (rfc 2396). IETF Request for Comments, Agosto de 1998.

[43] Weiser, M. Some Computer Science issues in Ubiquitous Computing. *Communications of the ACM,* Julho de 1993.

[44] WinZip Computing, Inc. *AES Encryption Information: Encryption Specification AE-2,* Janeiro de 2004. Disponível na internet, última visita em outubro de 2004. http://www.winzip.com/aes_info.htm#encryption.

[45] Bray, T.; Paoli, J.; Sperberg-McQueen, C.M. et al. *Extensible Markup Language (XML) 1.0*, Fevereiro de 2004. Disponível na Internet, último acesso em maio de 2004.

[46] Wright, C. P.; Martino, M. C.; Zadock, E. NCryptFS: A secure and Convenient Cryptographic File System. In: USENIX Technical Conference 2003. *General Track of the USENIX 2003 Annual Technical Conference*.

[47] Wright, C. P.; Dave, J.; Zadok, E. Cryptographic File Systems Performance: What You Don't Know Can Hurt You. In: 2003 IEEE Security in Storage Workshop (SISW 2003). *Proceedings*. 2003.

[48] Zadok, E.; Badulescu, I.; Shender, A. CryptFS: A stackable Vnode Level Encryption File System. Columbia U-CS Tech Report: Junho 1998.

[49] Freier, A. O.; Karlton, P.; Kocher, P.C. *The SSL Protocol Version 3.0,* Novembro 1996. Disponível na Internet, último acesso em setembro de 2004. http://wp.nestcape.com/eng/ssl3/draft302.txt