UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ DE RIBAMAR MARTINS BRINGEL FILHO

FRAMESEC: UM FRAMEWORK PARA A PROVISÃO DE SEGURANÇA FIM-A-FIM PARA APLICAÇÕES NO AMBIENTE DE COMPUTAÇÃO MÓVEL

FORTALEZA
2004

JOSÉ DE RIBAMAR MARTINS BRINGEL FILHO

FRAMESEC: UM FRAMEWORK PARA A PROVISÃO DE SEGURANÇA FIM-A-FIM PARA APLICAÇÕES NO AMBIENTE DE COMPUTAÇÃO MÓVEL

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade.

FORTALEZA
2004

JOSÉ DE RIBAMAR MARTINS BRINGEL FILHO

FRAMESEC: UM FRAMEWORK PARA A PROVISÃO DE SEGURANÇA FIM-A-FIM PARA APLICAÇÕES NO AMBIENTE DE COMPUTAÇÃO MÓVEL

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Aprovado em ___/___/_____

BANCA EXAMINADORA

_____

Profa. Dra. Rossana Maria de Castro Andrade (Orientadora)
Universidade Federal do Ceará – UFC

_____

Prof. Dr. Djamel F. H. Sadok
Universidade Federal de Pernambuco - UFPE

_____

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC

_____

Prof. Dr. André Jalles Monteiro
Universidade Federal do Ceará - UFC

Dedico esta dissertação

A Deus, por que Dele, e por Ele, e para Ele são
todas as coisas (Rm, 11:36).

A minha querida avó Maria Luiza (in memoriam),
fonte eterna de inspiração,
aos meus Pais, José Bringel e Maria do Socorro,
exemplos de honestidade e trabalho,
e aos meus irmãos, Eloísa, Enoque e Erick,
pela união incondicional e pela crença que essas pessoas
sempre tiveram na minha capacidade de aprendizado e luta.

AGRADECIMENTOS

Tentarei listar todas as "classes" e "instâncias" que compõem o *Framework* de ajuda começo-a-fim para o desenvolvimento deste trabalho, pois entendo que o mais importante não é o final alcançado, e sim todo o percurso realizado até aqui. À super-classe Pais, nas instâncias Socorro Bringel e José Bringel, por todo apoio desempenhado durante esta dura jornada, principalmente por executar o método construtor da Classe filho, criando a instância Bringel (será que já existia orientação a objeto?). Às instâncias da classe Irmão Eloísa, Enoque e Erick, por permanecerem de forma incondicional ao meu lado e, sempre que possível, executado o método "AjudarBringel(R$ Valor)". Em especial ao Erick, por ter compartilhado boa parte dessa jornada, dividindo os momentos difíceis e proporcionando alegrias, além de ajudar nas tarefas diárias (principalmente o método "OrganizarCasa()" e "LavarLouça()"). Aos Tios Constâncio, Mirtes, Jesus e Luiz Carlos, os quais compartilharam sua casa e atenção, sendo praticamente extensões da super-classe Pais. Aos primos Breno, Patrícia, Rafael, Luiza, Mirna e Míria, pelos momentos em família proporcionados, facilitando a migração de código da plataforma Teresina para Fortaleza. À Luciana, por ter permanecido ao meu lado (do outro lado da linha telefônica) durante todo o mestrado, e compreendido os momentos difíceis que passei, principalmente pela falta de dinheiro e tempo para executar o método "ViajarParaTeresina". À super-classe Orientadora, na instância da Profa. Rossana Andrade, por disponibizar todos os métodos necessários para o desenvolvimento de um bom trabalho, tais como "ConseguirBolsa", "DarApoio", "EntenderProblemas" e "AceitarDesculpas". Às vezes a execução destes métodos geravam a exceção "AVC", mas no final não deixava qualquer seqüela. Mais uma vez, executo o método "AgradecoPorTudo" e espero que os métodos continuem visíveis à minha classe. Ao professor André Jalles, por ter desempenhado o papel da classe Coorientador, disponibilizando suas tardes para realizar análises. As instâncias de financiadores, Instituto Atlântico e CNPQ, por investir durante dois anos na minha formação. Pena que às vezes ocorriam atrasos na execução do método "Depositar", o que ocasionava instabilidades na execução do *Framework*. Ao ambiente de execução "Embaixada Piauiense", localizado na Av. Santos Dumont, por ter me recebido e aturado por um período de 8 meses (quase uma gestação). Principalmente às instâncias da Classe "Chico" Daniel, Lineu e Fernando, por terem compartilhado esse curto e bom

período. Aos amigos de faculdade: William, Tarciane, Windson, Katy, Robson, Carlos, Rafael, Misael, Fabrício, Edgar, Moacyr e Suzana, que direta ou indiretamente contribuíram com a companhia bem como para o desenvolvimento deste trabalho em si. Ao amigo Hermano, pelos conselhos e a grande ajuda ao emprestar este computador que, sem ele, eu não teria nem estaria escrevendo neste momento. Enfim, agradeço a todos aqueles que disponibilizaram métodos à minha classe e perderam tempo nos seus ciclos de vida durante todo o período de desenvolvimento deste trabalho.

"Não se preocupe em entender,
viver ultrapassa todo entendimento"

Clarice Lispector

RESUMO

Dispositivos de computação móvel oferecem conectividade de rede através de sistemas de comunicação sem fio que possibilitam às aplicações obterem acesso a informações a qualquer momento e em qualquer lugar. Ao mesmo tempo em que possuem esta facilidade de conectividade, aplicações que realizam transações com dados sigilosos, tais como comércio móvel, requerem um alto nível de segurança. Entretanto, o meio sem fio é mais vulnerável a ataques passivos e ativos do que os meios de transmissão utilizados em redes tradicionais. Apesar dos sistemas de comunicação sem fio implementarem mecanismos de proteção para a transmissão de dados, eles apresentam vulnerabilidades e não garantem a segurança fim-a-fim desejada às aplicações. Para diminuir essas limitações, é necessário adicionar mecanismos de segurança nas camadas de rede, de transporte ou de aplicação. No entanto, os protocolos de segurança existentes que operam nas camadas de rede e de transporte, quando também não apresentam vulnerabilidades, são inviáveis de serem utilizados por dispositivos que possuem limitações de recursos. Deste modo, surge à necessidade de adicionar esses mecanismos na camada de aplicação. Este trabalho então propõe um *framework* de aplicação que provê reutilização da estrutura e da implementação de mecanismos de segurança para a provisão de segurança fim-a-fim às aplicações no ambiente de computação móvel. Além do *framework*, propõe ainda uma ferramenta que permite avaliar o desempenho dos algoritmos criptográficos que irão compor os pontos de flexibilização das instâncias do *framework*, utilizadas na construção de aplicações para um determinado dispositivo móvel analisado pela ferramenta. Portanto, os resultados das avaliações fornecem subsídios que ajudaram o desenvolvedor na definição dos algoritmos que serão utilizados nas instâncias do *framework*. Por fim, um estudo de caso que demonstra a utilização do *framework* combinado à ferramenta para a construção de aplicações seguras no ambiente de computação móvel também é introduzido neste trabalho.

Palavras-chave: computação móvel, segurança, criptografia, dispositivos móveis, framework.

ABSTRACT

Mobile computing devices offer network connectivity using wireless communication systems that make possible to applications to get information access at any time and any place. While it is easy to have connectivity, applications that have transactions with sensitive data, such as mobile commerce, require a high level of security. Wireless media are more vulnerable to passive and active attacks than the ones used in traditional networks. Despite wireless communication systems have implemented protection mechanisms for data transmission, they present vulnerabilities and no guarantee of end-to-end security to the applications. To reduce these limitations, it is necessary to add security mechanisms in the application, transport, or network layers. However, existing security protocols that operate in the transport and network layers also present vulnerabilities and they are impracticable to be used for devices that have resource limitations. Thus, there is a need to add these mechanisms in the application layer and this work proposes a framework that provides the reuse of a structure and an implementation of security mechanisms for the provision of end-to-end security to the applications in the mobile computing environment. Moreover, this master thesis presents a tool that allows of evaluating the performance of cryptographic APIs, implemented in platform J2ME, in the mobile devices. The use of the tool, combined to the framework, allows the construction of optimized security mechanisms, in accordance with performance factors in the device where the application is going to be executed. Finally, a case study, which demonstrates the use of the framework combined to the tool for the construction of safe applications in the environment of mobile computing, is also introduced in this work.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
Ciphertext	Texto encriptado
DM	Dispositivos Móvel
GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile Communications*
IEEE	*Institute of Electrical and Electronic Engineers*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
J2ME	*Java 2 Micro Edition*
LAN	*Local Area Network*
OSI	*Open Systems Interconection*
PA	Ponto de Acesso
PEARL	*Performance EvaluAtor of criptogRaphic aLgorithms for Mobile Devices*
Plaintext	Texto em plano, sem criptografia
SA	Servidor de Aplicação
SC	*Sample Collector*
SSL	*Secure Socket Layer*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UMTS	*Universal Mobile Telecommunication System*
WAP	*Wireless Application Protocol*
WLAN	*Wireless Local Area Network*
WTLS	*Wireless Transport Layer Security*

ÍNDICE

1 Introdução

Esta dissertação apresenta a proposta de um *framework* para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel. Além do *framework*, propõe uma ferramenta que permite avaliar o desempenho dos algoritmos criptográficos que irão compor os pontos de flexibilização das instâncias do *framework*, utilizadas na construção de aplicações para um determinado dispositivo móvel analisado pela ferramenta. Portanto, os resultados das avaliações fornecem subsídios que ajudaram o desenvolvedor na definição dos algoritmos que serão utilizados na instância do *framework*.

Neste capítulo, serão apresentadas a justificativa e a motivação para o desenvolvimento destas propostas, assim como os objetivos e as contribuições que se pretende alcançar. Ao final do capítulo, será descrito como está organizada o restante desta dissertação.

1.1 Caracterização do Problema e Motivação

Uma enorme diversidade de dispositivos móveis tem surgido no mercado com possibilidade de conexão a redes IP utilizando meios sem fio [4], por exemplo, celulares com suporte a WAP, celulares GPRS, celulares i-Mode (no Japão), notebooks, Palms e Pocket PCs com Bluetooth e IEEE 802.11.

A possibilidade de mudança de terminais de acesso fixo para dispositivos móveis impõe novos desafios no desenvolvimento de aplicações [5]. A capacidade de mobilidade e de armazenamento, o baixo poder de processamento, o tamanho compacto e o suporte aos sistemas de comunicação sem fio que permitem a comunicação com outros dispositivos e computadores conectados à Internet são exemplos desses desafios. Entretanto, a utilização dos sistemas de comunicação sem fio para o estabelecimento da comunicação e a facilidade crescente para disponibilizar informações acarretam sérios riscos à segurança. Infelizmente, esses dispositivos oferecem mecanismos de segurança muito aquém dos oferecidos nas redes fixas.

Os dispositivos móveis geralmente utilizam um dos sistemas existentes de comunicação sem fio (e.g., IEEE 802.11, GSM/GPRS, UMTS, Bluetooth) para realizar a transmissão dos dados ao servidor das aplicações, localizada na rede fixa. Transações

presentes nestas aplicações que operam com dados sigilosos necessitam que os mesmos sejam transmitidos de forma segura, o que é conseguido através do estabelecimento de um canal seguro fim-a-fim, do dispositivo móvel ao servidor da aplicação. Entretanto, como mencionado anteriormente, os mecanismos presentes nos sistemas de comunicação sem fio para o provimento de segurança ainda apresentam vulnerabilidades que permitem a ocorrência de ataques ativos e passivos. Além disso, estes mecanismos estão limitados a proteger apenas o meio sem fio, não provendo a segurança fim-a-fim desejada às aplicações. Portanto, faz-se necessário adicionar mecanismos às camadas superiores (e.g., de rede, de transporte e de aplicação) para a provisão de segurança.

Por um lado, os protocolos de segurança existentes que operam nas camadas de rede e de transporte, quando não apresentam vulnerabilidades, são inviáveis de serem utilizados por dispositivos móveis que possuem limitações de recursos. Sendo assim, os mecanismos devem ser adicionados na camada de aplicação para a provisão de segurança fim-a-fim às aplicações para dispositivos móveis.

Por outro lado, os mecanismos também devem ser construídos de acordo com os requisitos de segurança exigidos pela aplicação, que visam atingir os objetivos da criptografia, tais como confidencialidade, integridade, autenticidade e não-repudiação. Existem vários algoritmos criptográficos disponíveis (e.g., DES, Triple-DES) para cada um dos objetivos citados, que podem ser implementados ou apenas reutilizados, se estes se encontrarem disponíveis para a plataforma de desenvolvimento de aplicações (e.g., Personal Java, J2ME, SuperWaba, C) suportada pelo dispositivo móvel. A construção desses mecanismos é uma tarefa difícil, principalmente quando realizada por desenvolvedores de aplicações leigos em segurança da informação. A dificuldade é agravada, por que, geralmente, os requisitos de segurança não ocorrem de modo isolado, sendo necessário construir tais mecanismos de forma a atender combinações de requisitos distintos. Por exemplo, uma aplicação requer a confidencialidade da informação, bem como a integridade dos dados.

Este trabalho é motivado, portanto, pela necessidade de fornecer aos desenvolvedores de aplicações uma estrutura que permita a reutilização dos mecanismos construídos para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel.

1.2 Objetivos Específicos e Contribuições do Trabalho

Esta dissertação apresenta uma proposta de um *framework*, denominado de Framesec, para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel, bem como uma ferramenta, PEARL (*Performance EvaluAtor of criptogRaphic aLgorithms for Mobile Devices*), que permite avaliar o desempenho de algoritmos criptográficos implementados na plataforma J2ME (*Java 2 Micro Edition* [71]), que irão compor os pontos de flexibilização das instâncias do *Framesec*.

A proposta do Framesec consiste da especificação e da construção de blocos semifinalizados com propósitos de segurança, que podem ser instanciados e especializados para a construção de mecanismos de segurança a serem incorporados às aplicações que possuem requisitos de segurança no ambiente de computação móvel.

O principal objetivo da ferramenta PEARL é permitir ao desenvolvedor avaliar o desempenho dos algoritmos criptográficos que irão compor os pontos de flexibilização do *framework* no dispositivo móvel para o qual será instanciado, possibilitando a construção de mecanismos de segurança otimizados de acordo com os parâmetros de desempenho definidos pela ferramenta. Ela permite avaliar o desempenho de APIs criptográficas desenvolvidas na plataforma J2ME, podendo ser utilizada em uma vasta classe de dispositivos móveis, tais como Palms, Pockets PC e telefones móveis.

Em resumo, as principais contribuições deste trabalho são: a proposta do Framesec para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel; a ferramenta PEARL, que permite avaliar o desempenho de APIs criptográficas que irão compor os pontos de flexibilização do *Framesec*, desenvolvidas na plataforma J2ME, em uma vasta classe de dispositivos móveis; e o estudo de caso, que demonstra a integração entre o Framesec e PEARL.

1.3 Organização da Dissertação

Os próximos capítulos desta dissertação estão estruturados da seguinte forma:

- **Capítulo 2:** descreve os sistemas de comunicação sem fio que podem ser utilizados no contexto de computação móvel, onde são apresentadas as

vulnerabilidades presentes nos mecanismos de segurança oferecidos por esses sistemas para prevenir a ocorrência de ataques ativos e passivos;

- **Capítulo 3:** apresenta as soluções existentes para garantir a segurança fim-a-fim no ambiente de computação móvel, bem como a justificativa para o desenvolvimento do Framesec;
- **Capítulo 4:** descreve a proposta do Framesec;
- **Capítulo 5:** apresenta a ferramenta PEARL proposta neste trabalho;
- **Capítulo 6:** apresenta o estudo de caso que utiliza o Framesec bem como a ferramenta PEARL, demonstrando a integração entre as duas propostas para a construção de aplicações seguras para dispositivos móveis;
- **Capítulo 7:** apresenta as considerações finais do trabalho e direcionamentos para trabalhos futuros;
- **Apêndice A:** apresenta uma visão geral dos riscos de segurança a que estão sujeitos os dispositivos móveis;
- **Apêndice B:** descreve todos os algoritmos utilizados na ferramenta PEARL bem como nas instâncias do Framesec.

2 Aspectos de Segurança dos Sistemas de Comunicação sem Fio

Neste capítulo serão discutidos os aspectos de segurança dos sistemas de comunicação sem fio que podem ser utilizados pelos dispositivos móveis para obter acesso à rede fixa (e.g., Internet). Para cada sistema, serão apresentadas as vulnerabilidades dos mecanismos de segurança para a provisão de segurança para aplicações no ambiente de computação móvel.

2.1 Introdução

Antes de discutir os sistemas de comunicação sem fio e seus aspectos de segurança, será apresentado o ambiente de computação móvel considerado neste trabalho. Neste ambiente, ilustrado através da Figura 2.1, estão envolvidas as seguintes entidades: dispositivos móveis, sistemas de comunicação sem fio e servidor da aplicação. As aplicações hospedadas nos dispositivos móveis trocam informações com o servidor da aplicação utilizando uma conexão de dados, estabelecida através de um dos sistemas de comunicação sem fio (e.g., 802.11, GSM/GPRS) considerados neste trabalho.

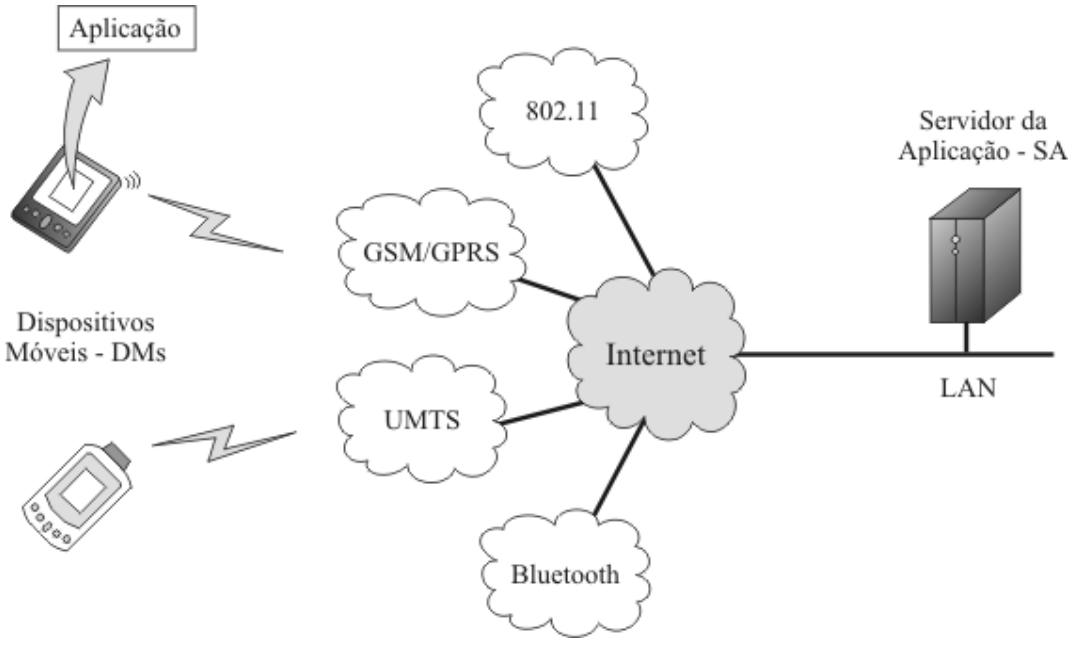

Figura 2.1. Ambiente de computação móvel considerado neste trabalho.

Os dispositivos móveis (DM) considerados neste trabalho podem ser classificados em:

- Telefone móvel: deve apresentar suporte a uma das plataformas de desenvolvimento de aplicações, tais como J2ME e BREW;
- PDA (Personal Digital Assistant);
- *Smart phone*: dispositivo que combina funcionalidades de telefone móvel com tecnologia de PDA.

As características inerentes a cada dispositivo podem atuar como limitantes aos tipos de serviços que estarão disponíveis aos usuários, tais como:

- Tamanho e quantidade de cores suportadas pelo *display*;
- Suporte a dispositivos de entrada de dados (e.g., *touch screen*), disponibilidade de teclado e mouse;
- Memória, CPU, poder de processamento;
- Conectividade de rede;
- Sistema operacional suportado (e.g., Palm OS, Microsoft Pocket PC, Symbiam OS);
- Disponibilidade de leitor de *smartcard* interno (e.g., para módulos SIM - S*ubscriber Identity Module*, em telefones móveis);
- Plataforma de desenvolvimento suportada (e.g., J2ME, Superwaba, BREW).

Por exemplo, a largura de banda do sistema de comunicação sem fio suportado pelo dispositivo pode restringir o acesso a serviços que demandam maiores taxas de transferência (e.g., aplicações de vídeo em tempo real).

Além disso, as aplicações hospedadas nos dispositivos móveis que realizam transações com dados sigilosos, requerem que os seguintes critérios de segurança sejam obedecidos:

- **Confidencialidade:** as informações transmitidas estejam acessíveis somente às duas partes envolvidas na transação (i.e., a aplicação no dispositivo móvel e o servidor da aplicação localizado na Internet ou rede fixa);
- **Integridade:** as informações transmitidas devem ser entregues ao seu destino sem qualquer ocorrência de modificação ou corrupção durante todo o caminho percorrido;
- **Autenticação:** as identidades de ambas as partes envolvidas na transação necessitam ser verificadas e confirmadas. As partes precisam provar, sem qualquer dúvida, a identidade de quem reivindica ser;

– **Não-repudiação:** nenhuma das partes envolvidas na comunicação pode repudiar o seu envolvimento. Ou seja, não deve ser possível para uma ou outra parte negar sua participação na comunicação.

As aplicações requerem ainda que a segurança seja garantida fim-a-fim, ou seja, da aplicação no dispositivo móvel ao servidor da aplicação, em oposição à segurança enlace-a-enlace. A segurança fim-a-fim corresponde à garantia de que os critérios acima mencionados serão cumpridos durante todo o tempo em que durar a transmissão, de forma independente da quantidade de nós de enlaces percorridos pelas informações, até o seu destino final. Na segurança fim-a-fim, o não cumprimento dos requisitos em um dos nós intermediários da rede comprometerá toda a segurança da comunicação, como será visto no Capítulo 3. Por sua vez, a segurança enlace-a-enlace se refere à segurança entre dois nós da rede, sendo esta inadequada ao ambiente de computação móvel considerado já que não oferece a segurança fim-a-fim desejada às aplicações.

Além dos requisitos mencionados acima, é necessário ainda proteger os dispositivos móveis, pois estes são alvos de ataques, tais como de acesso físico a dados e a vírus. O Apêndice A apresenta maiores detalhes sobre estes e outros tipos de ataques.

O servidor da aplicação também é alvo de ataques, que podem ser originados por usuários da rede local ou advindos da Internet. Neste caso, os riscos e medidas preventivas de segurança a serem adotadas no servidor da aplicação fogem do escopo deste trabalho.

Por fim, os sistemas de comunicação sem fio oferecem mecanismos de segurança com o objetivo de proteger o enlace sem fio compreendido entre os dispositivos móveis e a rede. Portanto, oferecem apenas segurança no nível de enlace, não provendo a segurança fim-a-fim desejada às aplicações. Serão apresentados nas próximas seções, detalhes dos sistemas de comunicação sem fio IEEE 802.11, GSM/GPRS, UMTS e Bluetooth, juntamente com os aspectos de segurança inerentes a cada sistema. Os sistemas acima foram considerados por serem amplamente difundidos e utilizados no contexto de computação móvel considerado neste trabalho

2.2 Padrão IEEE 802.11

O padrão de redes sem fio IEEE 802.11 [14] foi definido em 1997, permitindo alcançar velocidades de transmissão entre 1 e 2 Mbps. Em 1999 foi definido o padrão 802.11b, atingindo velocidades de até 11 Mbps.

Recentemente foram definidos os padrões 802.11a e 802.11g, ambos para velocidades de até 54 Mbps. Em paralelo, os principais fabricantes da área formaram a WECA (*Wireless Ethernet Compatibility Alliance*), com o objetivo de certificar a interoperabilidade entre produtos, bem como garantir a compatibilidade dos mesmos com o padrão 802.11b. Dispositivos móveis, tais como o iPAC, oferecem suporte nativo ao padrão 802.11b, sendo este entre os padrões 802.11 o mais comumente utilizado no contexto da computação móvel atual.

O padrão IEEE 802.11 está focado na definição das duas primeiras camadas do modelo OSI (i.e., camadas física e de enlace). Dois tipos de equipamentos são definidos nas redes sem fio 802.11: estação (*peer*), que pode ser um computador ou dispositivo móvel equipado com placa de rede sem fio; e ponto de acesso (PA), que atua como ponte entre as redes sem fio e fixa. Nas redes sem fio, duas topologias de comunicação são definidas: *ad hoc mode*, onde as estações comunicam-se diretamente entre si, e *infraestructure mode*, onde existe a presença de um PA intermediando as comunicações e fornecendo acesso à rede fixa.

A segurança no padrão IEEE 802.11 é baseada na autenticação e privacidade, podendo operar em dois modos: *Open System*, que realiza somente autenticação; e *Shared Key*, que realiza autenticação e privacidade. Este último utiliza o protocolo WEP (*Wired Equivalent Privacy*) como mecanismo de privacidade. A próxima seção descreve em detalhes os aspectos de segurança do padrão. Em seguida, serão apresentados cenários de utilização do padrão 802.11 no contexto da computação móvel, bem como as vulnerabilidades presentes nestes cenários.

2.2.1 Aspectos de Segurança

A segurança definida no padrão IEEE 802.11 envolve autenticação e privacidade na camada de enlace. As duas formas de autenticação citadas anteriormente (*Open System* ou *Shared Key*) podem ser utilizadas tanto para autenticação entre os dispositivos

móveis e PA (rede infra estruturada) quanto para autenticação entre os dispositivos em redes *ad hoc*.

A autenticação *Open System*, também chamada de autenticação nula, é a forma de autenticação mais simples e é utilizada por padrão nas redes IEEE 802.11. Nesse tipo de autenticação, todos os dispositivos móveis que realizarem a solicitação de autenticação ao PA ou ao dispositivo móvel comunicante serão autenticados, exceto em casos onde o dispositivo móvel ou o PA se recusa a autenticar algum outro em particular. O mecanismo envolve dois passos:

- 1º passo: a estação solicitante declara sua identidade e solicita autenticação;
- 2º passo: a estação solicitada informa o resultado da autenticação. Se o resultado da autenticação for positivo (*successful*), as estações estarão mutuamente autenticadas.

A autenticação *Shared Key* envolve o compartilhamento de uma chave secreta entre os dispositivos móveis (redes *ad hoc*), ou entre o dispositivo móvel e o PA em redes infra-estruturas. A chave secreta deve ser entregue aos nós participantes (DM e PA ou dois DMs) através de um canal seguro independente do IEEE 802.11. No modo *Shared Key*, o nó que inicia a autenticação é referenciado como *requester* e o outro *responder* [14]. Esta forma de autenticação envolve quatro passos:

- 1º Passo: *requester* envia uma mensagem {*authentication request*} ao *responder* solicitando autenticação por *shared key*;
- 2º Passo: *responder* envia uma mensagem de resposta {*authentication response*} contendo um desafio (*challenge*);
- 3º Passo: *requester* cifra o desafio com sua chave WEP e o devolve em uma nova mensagem {*authentication request*};
- 4º Passo: Se o *responder* decifrar o *authentication request* e obtiver o desafio original, ele responde com um *authentication response* concedendo acesso ao *requester*.

O protocolo WEP visa fornecer às redes sem fio privacidade equivalente à das redes com fios [14]. Ele utiliza o algoritmo *stream cipher* RC4 para garantir a privacidade, que utiliza chave criptográfica de 64 bits, composta por 40 bits da chave propriamente dita e um vetor de inicialização IV (*Initialization Vector*) público de 24 bits. O RC4 possui falhas no algoritmo de escalonamento de chaves (*Key Scheduling Algorithm*), conforme citado em [18] e [19]. Além das falhas presentes no algoritmo RC4 que compõe o protocolo WEP, foram observadas recentemente outras falhas nesse

protocolo [24][25][26], tais como repetição do IV, reuso de chaves e fragilidade no mecanismo de integridade.

Para aumentar a segurança do padrão 802.11, a IEEE criou uma solução temporária chamada TKIP (*Temporal Key Integrity Protocol*), que acrescenta os seguintes elementos [26]:

- MIC (*Message Integrity Code*) para evitar fraudes;
- Seqüenciamento de pacotes, para prevenir ataques de *replay*;
- Construção de chaves criptográficas por pacote, para evitar ataques FMS (Fluhrer-Mantin-Shamir) [18]. Este ataque é realizado para descobrir o fluxo de chaves do algoritmo (que no caso é o RC4) a partir do texto cifrado (*ciphertext*) e do texto plano (*plaintext*).

Como solução de longo prazo, a IEEE está desenvolvendo o CCMP (*Counter-Mode-CBC-MAC Protocol*) [26], também com o objetivo de resolver as deficiências do WEP. Este novo protocolo utiliza o algoritmo de criptografia AES (*Advanced Encryption System*, descrito no Apêndice B) que é incompatível com o WEP, pois este utiliza o algoritmo RC4.

2.2.2 Cenários de Utilização

No modo de operação padrão (i.e., sem qualquer configuração de segurança realizada por parte do administrador de rede), as redes 802.11 não oferecem quaisquer mecanismos de segurança, o que permite a ocorrência de escutas promíscuas e a manipulação das informações que trafegam no meio sem fio. Para prover certo nível de segurança às WLANs, deve-se utilizar o protocolo WEP, conforme mencionado na Seção 2.2.1. O protocolo WEP pode ser habilitado pelo administrador da rede, o que adiciona proteção aos dados que trafegam no meio sem fio compreendido entre os DMs e o PA, no caso das redes infra estruturadas.

A Figura 2.2 ilustra dois cenários de utilização das redes sem fio 802.11 infra-estruturadas no contexto da computação móvel, permitindo a comunicação entre os dispositivos móveis (DMs) e o servidor da aplicação (SA). O primeiro cenário compreende a comunicação do DM1 com o servidor da aplicação através do PA1. Por sua vez, o segundo cenário corresponde à comunicação entre o DM2 e o servidor da aplicação através do PA2.

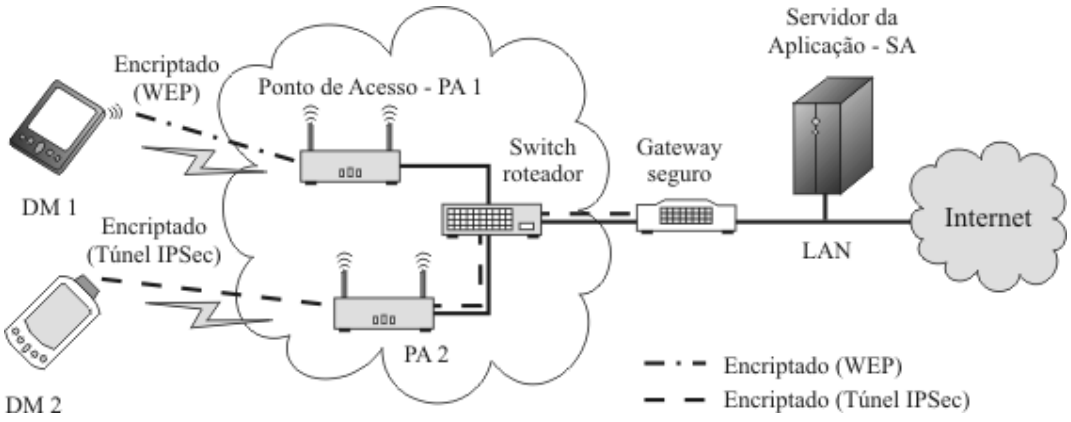

Figura 2.2. Comunicação entre os DMs e o SA utilizando uma rede sem fio 802.11.

Analisando o primeiro cenário, observa-se que o meio sem fio, compreendido entre o DM1 e o PA1, está protegido através da criptografia dos dados realizada pelo protocolo WEP. Entretanto, o trecho de rede compreendido entre o PA1 e o SA está desprotegido, permitindo a qualquer usuário da rede ou intruso advindo da Internet recuperar informações através de escuta promíscua.

Apesar da utilização do protocolo WEP para proteger o meio sem fio, os riscos contra a confidencialidade e integridade de dados (citados na Seção 2.1) continuam existindo, já que este apresenta falhas conforme discutido na Seção 2.2.1. Portanto, mesmo considerando o meio sem fio seguro no primeiro cenário ilustrado na Figura 2.2, a adoção do WEP não garante a segurança fim-afim desejada aos dados das aplicações, já que o trecho de rede compreendido entre o PA1 e o SA continua desprotegido. Sendo assim, a proteção do meio sem fio não oferece garantias de segurança fim-a-fim que são desejadas às aplicações que executam transações com dados sigilosos.

Em vista às falhas no WEP e a inexistência de segurança fim-a-fim, é necessário adicionar mecanismos de segurança às camadas superiores (e.g., de rede, transporte ou aplicação). Uma abordagem possível é a utilização de redes privadas virtuais de dados (*Virtual Private Network* – VPN), em particular o uso do protocolo IPsec (*Internet Protocol Security*) [7], com o objetivo de estabelecer segurança na camada de rede. O IPsec (mais especificamente o *ESP Tunnel Protocol*) é um padrão Internet definido com o fim de proteger pacotes IP transmitidos entre dois nós (e.g., na Figura 2.2, proteger os pacotes IP na comunicação entre o DM2 e o *Gateway*). Note que informações da camada de enlace (e.g., o endereço MAC) continuam desprotegidas.

A utilização do IPSec se torna inviável em comunicações envolvendo dispositivos móveis que possuem limitações de memória e processamento, principalmente por apresentar um alto custo computacional em decorrência da sua complexidade [10]. Outro fato a ser observado é que o túnel seguro para que a segurança seja garantida fim-a-fim deve ser estabelecido entre os DMs e o SA, o que não acontece. A Figura 2.2 ilustra um cenário onde a VPN é estabelecida entre o DM2 e o *gateway*, não oferecendo a segurança fim-a-fim desejadas às aplicações, já que o trecho compreendido entre o *gateway* e o SA encontra-se desprotegido.

Em decorrência dos problemas apresentados acima, mecanismos de segurança devem ser adicionados às camadas superiores no IEEE 802.11 para a provisão de segurança fim-a-fim às aplicações no ambiente de computação móvel, foco deste trabalho.

2.3 GSM / GPRS

As redes GSM (*Global System for Mobile Communications*) surgiram da tentativa do *European Telecommunication Standards Institute* - ETSI padronizar os sistemas celulares privados que os países europeus vinham desenvolvendo na década de 1980. As redes GSM foram desenvolvidas para suportar qualidade de voz, baixo custo de operação, compatibilidade com ISDN (*Integrated Services Digital Network*) e "*roaming global*", sendo utilizada atualmente por mais de 200 países em todo o mundo [33][79].

No GSM, as estações móveis (*Mobile Stations*) são formadas pelo equipamento móvel (*Mobile Equipment*) e o módulo SIM (*Subscriber Identity Module*). O equipamento móvel possui o hardware (transmissor e o receptor) e o software responsáveis pela comunicação com a rede através da interface sem fio.

Nos primeiros anos de implantação do GSM (início dos anos 90), as estações móveis eram limitadas a possuir funcionalidades de um telefone convencional. Nessa época já era possível estabelecer sessões de dados sobre conexões de circuito comutado. Estas conexões, além de lentas (9,6 kbits/s) requeriam dispositivos computacionais separados (e.g., PDA) das estações móveis para a execução das aplicações reduzindo a mobilidade.

Com a popularização do uso da Internet e de outros serviços de dados em meados da década de 1990, previu-se que as redes GSM não seriam capazes de suportar a demanda por tráfego de dados. Em 1997 o ETSI publicou o modo de funcionamento

do GPRS na especificação da FASE 2+ do GSM. Surgiram então as redes GPRS (*General Packet Radio Service*), desenvolvidas para suportar serviços de dados, pois se baseiam na transmissão por comutação de pacotes, com taxa máxima de 171,2 kbps (em contraste com a limitada taxa do GSM de 9,6 kbps). Com a expansão do núcleo da rede GSM através da adição de novos elementos de serviços de dados, as estações móveis também se tornaram mais robustas. Surgiram então os seguintes serviços:

- SMS (*Short Message Service*): permite a troca de mensagens curtas (160 caracteres) sobre o canal de sinalização;
- WAP (*Wireless Application Protocol*): possibilita acesso a conteúdo da Internet em formato WML (*Wireless Mark-up Language*);
- HSCSD (*High Speed Circuit Switched Data*): permite altas taxas de transferência de dados através do empacotamento de canais;
- GPRS (*General Packet Radio Service*): estende a rede GSM, conforme citado anteriormente, com serviços orientados a pacotes. Com o GPRS, os nós móveis podem permanecer sempre conectados, mesmo que não realize transmissão de dados, sem apresentar alocação de recursos da rede. O GPRS pode também ser utilizado como portador de serviços para WAP e SMS.

A Figura 2.3 ilustra a arquitetura básica das redes GSM/GPRS, incluindo os elementos das redes inteligentes (INs - *Intelligent Networks*) e componentes do SMS. Nesta arquitetura, os dispositivos ou estações móveis se comunicam através da interface sem fio com a *Base Transceiver Station* (BTS), a qual é parte da *Base Station Subsystem* (BSS). As BTSs são controladas através da *Base Station Controller* (BSC), que está conectada ao *Mobile Switching Center* (MSC) e ao *Service GPRS Support Node* (SGSN). A MSC é responsável pelo estabelecimento das conexões de tráfego (i.e., dados e voz) com as BSCs, bem como com outras redes e MSCs. Já o SGSN é responsável pela entrega dos pacotes de dados originados ou destinados às estações móveis.

O SGSN, por sua vez, está conectado ao *Gateway GPRS Support Node* (GGSN), elemento de rede que atua como *gateway* com as redes externas (e.g., rede IP, Internet ou provedores de serviço de rede) realizando a entrega de dados, via roteadores, ao destino apropriado (e.g., o SA, conforme a Figura 2.3).

Além dos elementos do núcleo da rede GSM/GPRS citados acima, existem os repositórios de dados *Home Location Register* (HLR), *Visitor Location Register* (VLR), *Authentication Center* (AuC) e os elementos de rede *Service Control Point*

(SCP), *Short Message Service Center* (SMSC) e *Gateway Mobile Services Switching Center* (GMSC), que atua como *gateway* com as *Public Switched Telecommunications Network* (PSTN) e *Integrated Service Digital Network* (ISDN). O HLR é um repositório que mantém informações das estações móveis cadastradas em uma área de serviço, já o VLR é um repositório temporário que mantém os dados das estações móveis que estão fora de sua área de serviço (i.e., em *roaming*). Cópias das chaves secretas associadas a cada módulo SIM, usadas no processo de autenticação da estação móvel e na encriptação do enlace estão armazenadas no repositório AuC. Na próxima seção serão tratados os aspectos de segurança das redes GSM/GPRS.

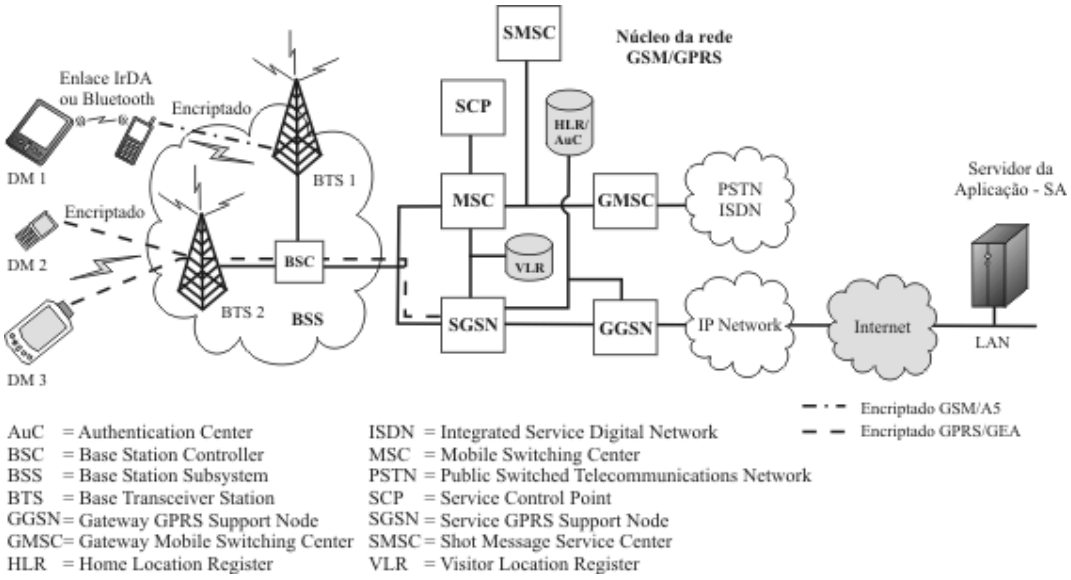

Figura 2.3. Comunicação entre os DMs e o SA utilizando uma rede GSM/GPRS.

2.3.1 Aspectos de Segurança

Nas redes GSM/GPRS, quando um usuário solicita o serviço de rede à operadora, o DM recebe um identificador único chamado *International Mobile Subscriver Identity* (IMSI), e uma chave de autenticação Ki. O IMSI e a chave Ki são armazenados no módulo SIM do DM. O SIM possui proteção por senha definida pelo usuário (*Personal Identification Number* - PIN). No lado da rede, o IMSI, a Ki e outras informações do usuário são armazenadas no HLR e no centro de autenticação AuC.

As redes GSM/GPRS oferecem os seguintes mecanismos de segurança ao enlace compreendido entre os DMs e a rede [80]:

- Confidencialidade do IMSI;
- Autenticação do IMSI;

– Confidencialidade dos dados dos usuários em conexões físicas;
– Confidencialidade dos dados de usuários sem conexão;
– Confidencialidade de elementos de informação de sinalização.

A autenticação do DM na rede ocorre da mesma forma, tanto nas redes GSM quanto nas GPRS. O processo é realizado através de um mecanismo de autenticação do tipo "desafio e resposta", que utiliza o protocolo A3, a chave secreta Ki e o vetor RAND (cada elemento do vetor corresponde a uma seqüência aleatória de 128 bits).

Nas redes GSM, o enlace sem fio compreendido entre o DM e a BTS, que corresponde à camada RR (*Radio Resorce Managent*), é encriptado utilizando o algoritmo *stream cipher* A5 e a chave Kc criada após o processo de autenticação. Nas redes GPRS, a encriptação se diferencia das redes GSM apenas quanto ao algoritmo utilizado, o GPRS *Encryption Algorithm* (GEA) [41] (versão do algoritmo A5 para as redes GPRS), e à extensão de rede onde é realizada. A encriptação dos dados nas redes GPRS compreende o trecho de rede entre o DM até o SGSN, que correponde à camada LLC (*Logical Link Control*).

O algoritmo *stream cipher* A5, que é do tipo caixa preta (i.e., não possui o código aberto), possui três versões: o A5/1, versão utilizada nos EUA e na Europa Ocidental; o A5/2, versão exportável e o A5/0, versão sem criptografia utilizada por países sob sanções da ONU.

A versão A5/1 [34][35] possui chave de 64 bits, sendo que os dez últimos bits não são utilizados, tendo, portanto, chave efetiva de 54 bits. Por ser um algoritmo do tipo caixa preta, a descoberta do seu funcionamento coloca em risco a segurança do sistema. Este algoritmo já foi descoberto por técnicas de engenharia reversa, sua implementação pode ser vista em [36]. A mesma equipe descobriu o funcionamento posteriormente do A5/2 [37], utilizando as mesmas técnicas de engenharia reversa. Os ataques às duas versões são apresentados em [39].

Em [40] são propostas melhorias no processo de autenticação nas redes GSM através do uso do algoritmo RC5, visando proteger o identificador do usuário bem como diminuir o *delay* da fase de autenticação.

Nas redes GPRS, todos os dados transferidos entre os DMs e o SGSN (i.e., camada LLC) são encriptados, conforme citado anteriormente. Durante o processo de autenticação, o SGSN pode decidir se utiliza ou não a encriptação. Caso utilize, será estabelecida a chave Kc entre o DM e o SGSN. O algoritmo GEA utilizado para a

encriptação está disponível em várias versões. Por exemplo, quando um DM GPRS tenta estabelecer uma conexão com uma rede estrangeira, será iniciada a negociação para se estabelecer à versão do GEA compatível com a rede e o DM. Caso nenhuma versão compatível com ambos seja encontrada, a conexão será estabelecida em sinal aberto (i.e., sem encriptação).

Uma vez que os algoritmos utilizados nas redes GPRS são mantidos em sigilo, ainda não existem, até o presente momento da escrita deste trabalho, ataques conhecidos e disponibilizados à comunidade científica.

Nas redes GSM/GPRS a chave Ki nunca é transmitida na rede, portanto, não é possível recuperá-la através da escuta promíscua do meio sem fio. Mesmo assim, as redes GSM/GPRS apresentam vulnerabilidades. Por exemplo, elas estão vulneráveis a ataques "*man-in-the-middle*" (uma entidade atacante se posiciona entre as partes comunicantes realizando a coleta, bem como a entrega, das informações de autenticação destas partes), já que a rede não é autenticada pelo DM permitindo a uma BTS intrusa interceptar e redirecionar o processo de autenticação a uma rede falsa. A mesma BTS pode ainda suprimir a confidencialidade do IMSI e este se tornar invisível ao DM, ocasionando um ataque de negação de serviço.

Além dos ataques acima citados, existem vulnerabilidades nos algoritmos de autenticação e encriptação do GSM, mas geralmente requerem acesso físico (em curto espaço de tempo) ao SIM. Outros ataques podem ainda serem visto em [39].

2.3.2 Cenários de Utilização

A Figura 2.3 ilustra os cenários de utilização das redes GSM/GPRS no ambiente de computação móvel considerado neste trabalho. São eles:

- *Cenário 1:* o DM1, que utiliza uma estação móvel (através de um enlace IrDA, Bluetooth ou cabo de dados) para obter acesso a rede GSM através de uma sessão de dados sobre uma conexão de circuito comutado, se comunica com o SA;
- *Cenário 2:* DM2 e DM3, que utilizam conexões de dados orientadas a pacotes (i.e., GPRS), comunicam-se com o SA.

No cenário 1, observa-se que o trecho de rede compreendido entre a BTS1 e o SA está desprotegido. Isso por que as redes GSM só oferecem encriptação de dados no enlace sem fio, correspondente à camada RR, favorecendo ataques contra a

confidencialidade dos dados transmitidos ao SA. Os ataques podem ser originados internamente na própria rede ou por usuários oriundos da Internet. Além disso, a operadora de telefonia móvel pode habilitar ou não a encriptação na camada RR. Caso não esteja habilitado, a transmissão de dados é realizada em sinal aberto. Outro problema ocorre no *roaming* entre redes de operadoras diferentes, onde a operadora estrangeira pode oferecer, ou não, a encriptação na camada RR.

Analisando o cenário 2, os dados transmitidos pelos DMs não estão encriptados além do SGSN. O tráfego de dados entre o SGSN e o SA permanece desprotegido, a menos que seja fornecido outro mecanismo de segurança ao trecho de rede. Por exemplo, poder-se-ia criar uma VPN (conforme citado na Seção 2.2.2) entre o GGSN e o SA, encriptando toda a comunicação de dados entre os nós. Outra opção seria utilizar uma linha dedicada de dados, a qual proveria segurança adicional e largura de banda constante, do GGSN ao SA. Porém, as duas soluções requerem configuração interna da rede da operadora, o que incide em custos adicionais e torna, em alguns casos, a utilização destas inviáveis.

Conclui-se, portanto, que as redes GSM/GPRS também não oferecem mecanismos de segurança fim-a-fim desejado às aplicações que realizam transações com dados sigilosos no ambiente de computação móvel, objetivo desta pesquisa.

2.4 UMTS

O UMTS (*Universal Mobile Telecommunication System*) é um sistema de telecomunicações móveis de terceira geração (3G) que está sendo desenvolvido dentro do *framework* definido pelo ITU (*International Telecommunications Union*), conhecido como IMT-2000 (*International Mobile Telecommunications - year 2000*). A principal diferença das redes GSM/GPRS está na tecnologia de rádio utilizada (UMTS *Terrestrial Radio Access Network* – UTRAN).

A rede UMTS é logicamente dividida em duas partes, o *Core Network* (CN) e o *Generic Radio Access Network* (GRAN) [20] [79]. O CN reutiliza vários elementos presentes nas redes GSM/GPRS e está dividido em dois domínios: *Circuit-Switched* (CS) e *Packet-Switched* (PS).

O domínio CS é composto por entidades que alocam recursos, de forma dedicada, no momento do estabelecimento de conexões para os tráfegos de usuários e de

sinalização de controle, e liberam esses recursos quando a sessão é finalizada. Geralmente as chamadas de voz são tratadas por essas entidades.

As entidades no domínio PS transportam os dados de usuário na forma de pacotes autônomos, que são roteados de forma independente uns dos outros; o transporte de dados orientado a pacotes elimina a limitação das redes 2G para a transmissão de dados de forma eficiente.

A rede UTRAN é uma implementação do conceito de GRAN. As funções executadas por seus componentes são:

- Gerenciamento de recursos de rádio;
- Controle de potência de *downlink* e *uplink*;
- Gerenciamento de *handover* (mudança de BTS sem a finalização da chamada) e de alocação de canais de transmissão.

Desde que vários componentes do CN são herdados das redes GSM/GPRS, o UMTS permite conexão com redes que utilizam a mesma tecnologia de rádio das redes GSM. Como conseqüência, a BSS GSM e o *Radio Network Subsystem* (RNS) UMTS podem coexistir dentro de uma mesma rede móvel pública UTRAN.

As redes UTRAN são compostas, além das entidades de rede comuns às redes GSM/GPRS (e.g., MSC, HLR, VLR, AuC, SGSN, GGSN), dos elementos *User Equipment/Mobile Station* (UE/MS), Node B e *Radio Network Controller* (RNC).

O UE/MS corresponde à estação móvel ou dispositivo móvel (DM), conforme já vem sendo utilizado neste trabalho, que consiste de um *Mobile Equipment* (ME) e de um *UMTS Subscriber Identity Module* (USIM). O USIM corresponde a um *IC Card* removível que opera em conjunto ao ME para prover acesso aos serviços 3G, que possuindo as seguintes funções: armazenar o identificador único de usuário, chaves de criptografia e informações relacionadas ao assinante; e executar operações de autenticação e encriptação de dados.

O Node B corresponde à BTS das redes GSM nas redes UTRAN, disponibilizando serviço de rádio a uma ou mais células da rede. As suas funções incluem: detecção de rede no canal de transporte e indicação às camadas superiores; modulação e demodulação de sinal de rádio; e medida e notificação às camadas superiores do aumento de potência do sinal. Atualmente é utilizado o WCDMA (*Wideband Code Division Multiple Access*) para a multiplexação do sinal de rádio entre o DM e o *Node B*.

Por último, o RNC é responsável pelas operações de gerência de recursos de rádio de cada *Node B* conectado a este. A Figura 2.4 ilustra arquitetura básica da rede UMTS, onde o RNC está conectado ao CS através da interface IuCS e ao PS através da interface IuPS. O RNC não somente controla os recursos de rádio do DM, como também o tráfego de e para os serviços de rede que estão sendo utilizados por este. As suas tarefas incluem: processamento do tráfego de voz e dados; *handover* entre células; e configuração e finalização de chamadas.

2.4.1 Aspectos de Segurança

As redes UMTS oferecem características de segurança no nível de enlace, através dos seguintes mecanismos:

- autenticação mútua;
- confidencialidade dos dados de usuário e de informações de controle de sinalização;
- integridade das informações de controle de sinalização.

Conforme descrito na Seção 2.3, os principais problema de segurança presentes nas redes GSM/GPRS são originados por dois fatores:

- autenticação em sentido único (o DM não autentica a rede, apenas é autenticado por esta);
- encriptação opcional.

Nas redes UMTS a autenticação mútua (i.e., o DM autentica a rede assim como é autenticado por esta) e a encriptação do tráfego obrigatória eliminam os problemas apresentados acima presentes nas redes GSM.

A autenticação mútua é realizada através do UMTS *Authentication and Key Agreement* (UMTS AKA), que consiste de um mecanismo de segurança responsável por funções de autenticação bem como do processo de estabelecimento de chaves de segurança. O UMTS AKA é um protocolo de autenticação do tipo "desafio/resposta", que visa manter o máximo de compatibilidade com o protocolo de estabelecimento de chaves e autenticação utilizado nas redes GSM/GPRS, o que torna mais fácil e transparente a transição entre as duas redes.

O UMTS AKA é executado após a ocorrência das seguintes situações:

- primeiro registro de um usuário na rede;
- requisição de serviço;

– requisição de atualização de localização;
– atendimento de requisição de ligação ou requisição de re-estabelecimento de conexão.

O mecanismo que fornece integridade das informações de controle de sinalização é o UMTS *Integrity Algorithm* (UIA), que é implementado tanto no DM (no USIM) quanto no RNC. Este mecanismo utiliza o algoritmo f9 [21], que realiza no DM o cálculo do código de autenticação de mensagem (chamado MAC-I) de 32 bits, utilizando como parâmetro de entrada as informações de sinalização (que corresponde ao campo MESSAGE), a chave de integridade compartilhada IK (*Integrity Key*) de 128 bits, um valor de 32 bits dependente do tempo (campo COUNT-I), um bit identificador da direção da transmissão chamado DIRECTION e um número randômico de 32 bits (nomeado de FRESH) gerado no RNC. O MAC-I é então concatenado às informações de sinalização e enviado à rede. O RNC, ao receber esses dados, realiza o cálculo do XMAC-I (que corresponde ao código de autenticação de mensagem MAC-I, sendo que este é calculado no lado da rede) a partir dos dados de sinalização recebidos, utilizando os mesmo parâmetros e o compara com o MAC-I calculado e enviado pelo DM. Desta forma, se o valor do MAC-I coincidir com o XMAC-I calculado, a integridade estará mantida.

Ao contrário do mecanismo de integridade, o mecanismo de confidencialidade opera tanto sobre as informações de sinalização quanto sobre os dados de usuários, protegendo o tráfego no enlace de rede entre o DM e o RNC. O algoritmo utilizado é o f8 [21], este recebe os seguintes parâmetros de entrada: a chave de criptografia CK (*Cipher Key*) de 128 bits, um valor dependente do tempo COUNT-C de 32 bits, um identificador de portador (chamado BEARER), a direção da transmissão DIRECTION e o tamanho do *key stream* (nomeado de LENGTH) que será gerado. O *key stream* gerado é utilizado posteriormente para encriptar o bloco de dados de entrada (PLAINTEXT), produzindo o bloco de saída (CIPHERTEXT).

Os algoritmos f8 e f9 utilizados, respectivamente, para garantir a confidencialidade e integridade nas redes UMTS, são construídos a partir do *block cipher* KASUMI [22], que é um algoritmo padrão utilizado nos sistemas 3GPP.

As redes UMTS oferecem, portanto, além da confidencialidade dos dados dos usuários e integridade das informações de sinalização, proteção contra ataques de *replay* através de números de seqüência adicionados ao vetor de autenticação. O UMTS introduz novos algoritmos de criptografia (f8 e f9) e utiliza longas chaves de

encriptação (128 bits). Desta forma, ainda não são conhecidas vulnerabilidades nos mecanismos de segurança utilizados pelo sistema.

2.4.2 Cenários de Utilização

Nas redes UMTS apenas um cenário de utilização será apresentado, que consiste na comunicação dos DM1 e DM2 com o SA utilizando uma conexão de dados estabelecida com a rede (observe a Figura 2.4). Neste cenário, o enlace sem fio entre os DMs e o *Node B* é encriptado utilizando o algoritmo f8, sendo que a encriptação é estendida até o RNC localizado no RNS, conforme citado na Seção 2.4.1. Entretanto, o trecho de rede compreendido entre o RNC e o SA realiza transmissão em aberto.

Portanto, a segurança dos dados transmitidos das aplicações hospedadas nos DMs só é garantida durante o seu percurso no enlace sem fio, não obedecendo os requisitos de segurança fim-a-fim descritos na Seção 2.1. Assim, existem riscos de segurança contra a confidencialide dos dados das aplicações, exigindo a adição de mecanismos de segurança às camadas superiores.

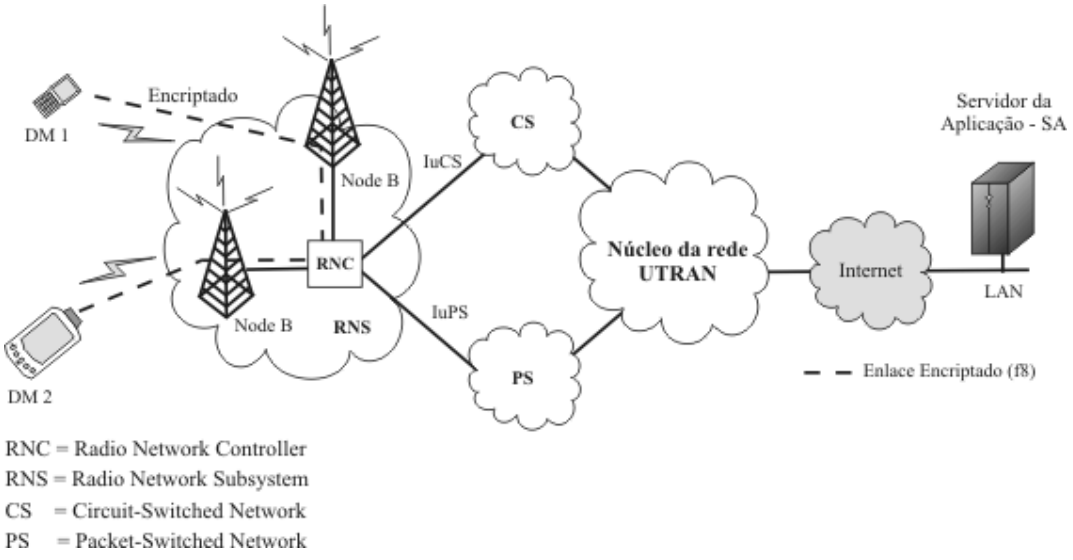

Figura 2.4. Comunicação entre os DMs e o SA utilizando o sistema UMTS.

2.5 Bluetooth

O Bluetooth é uma tecnologia de baixo custo, de simples configuração e baixo consumo de energia, que oferece conectividade *ad hoc* em redes sem fio de pequeno alcance

(WPANs), desenvolvida pelo *Bluetooth Special Interest Group* (SIG) [42]. O Bluetooth SIG lançou uma especificação industrial aberta, contendo duas partes: o núcleo e os perfis. O núcleo define as características do rádio e a pilha de protocolos para a comunicação entre os dispositivos. Os perfis especificam quais protocolos da pilha devem ser implementados para determinada aplicação.

O rádio Bluetooth opera na banda 2.4-GHz ISM (*Industrial, Scientific and Medical*) livre de licença e emprega a técnica de saltos de freqüência com espalhamento espectral (FHSS) na transmissão, podendo ser incorporada a equipamentos, tais como dispositivos móveis, impressoras, computadores e telefones móveis. O espectro de freqüência é dividido em 79 canais de rádio-freqüência (23 em alguns países), cada canal com 1 MHz de largura de banda e freqüência de saltos de 1600/s.

A camada física do Bluetooth é responsável, entre outras funções, pela formação das *piconets* e dos enlaces. A *piconet* é a unidade arquitetural básica de uma rede Bluetooth, podendo conter até oito integrantes ativos, onde os dispositivos compartilham o mesmo esquema de saltos de freqüência. Em cada *piconet*, um dos dispositivos assume o papel de mestre e os outros se comportam como escravos. O mestre de uma *piconet* é responsável por ditar a seqüência de saltos e a fase nesta seqüência. Os saltos de freqüência ocorrem a cada 625 microssegundos, a esse intervalo de tempo dá-se o nome de *slot* (ou *timeslot*).

Se dois ou mais dispositivos de *piconets* distintas quiserem se comunicar, pode haver a interconexão dessas redes, criando uma *scatternet* [42]. A *scatternet* se forma quando ao menos uma estação, chamada ponte, participa de duas ou mais *piconets*. Como normalmente as estações possuem apenas uma interface Bluetooth, a ponte não pode estar ativa em duas *piconets* simultaneamente. Por isso, ela divide, no tempo, sua participação nas múltiplas *piconets*, tarefa essa chamada de escalonamento *interpiconet*.

2.5.1 Aspectos de segurança

Cada dispositivo Bluetooth possui quatro entidades encarregadas da segurança no nível de enlace [42]:

- o *Bluetooth Device Address* (BD_ADDR): corresponde a um endereço de 48 bits único para cada dispositivo (equivalente a um endereço MAC para placas de rede *Ethernet*). Este endereço é definido pelo IEEE;

– Chave de Autenticação Privada (*Private Authentication Key*): consiste em um número pseudo-aleatório de 128 bits utilizado para propósitos de autenticação;
– Chave de Encriptação Privada (*Private Encryption Key*): tamanho variando de 8 a 128 bits;
– Número pseudo-aleatório (RAND) de 128 bits: gerado pelo próprio dispositivo Bluetooth.

O dispositivo utiliza o RAND para vários propósitos nos esquemas de encriptação e autenticação. Este número pode ser regenerado para formar diferentes números randômicos, quando o dispositivo assim achar necessário.

O Bluetooth provê três modos (perfis) de segurança gerais de acesso (*Bluetooth Generic Access Profile*): não-seguro, segurança no nível de serviço e segurança no nível de enlace [44]. A diferença entre os modos de segurança no nível de serviço e no nível de enlace é que, neste último, os dispositivos Bluetooth iniciam os procedimentos de segurança antes do estabelecimento do canal de comunicação.

Além dos modos de segurança, existem ainda diferentes níveis de acesso para os dispositivos e para os serviços utilizados. Para os dispositivos, existem dois níveis: dispositivo confiável (*trusted device*) e não confiável (*untrusted device*). O dispositivo confiável possui acesso irrestrito a todos os serviços. Para os serviços, existem os níveis: serviços que requerem autorização e autenticação, serviços que requerem somente autenticação e serviços abertos a todos os dispositivos.

O Bluetooth oferece, portanto, os seguintes mecanismos de segurança no nível de enlace:

– autenticação direcional ou mútua;
– autorização (de serviço ou de dispositivo);
– encriptação de dados.

Na especificação Bluetooth, todas as transações seguras entre duas ou mais entidades envolvidas utilizam as chamadas *link keys*. Uma *link key* é um número pseudo-aleatório de 126 bits utilizado tanto no processo de autenticação quanto no de encriptação, como um dos parâmetros para a derivação da chave de encriptação.

O tempo de vida de uma *link key* depende do seu tipo, podendo ser semipermanente ou temporária. No caso de uma chave temporária, o tempo de vida é equivalente à duração de uma sessão, não podendo ser reutilizada em sessões posteriores. Esta modalidade de chave é utilizada principalmente em sessões ponto-

multiponto, onde uma mesma informação é transmitida a vários destinatários. Já uma *link key* semipermanente pode ser utilizada após a finalização da sessão, sendo utilizada com propósitos de autenticação dos dispositivos Bluetooth que a compartilham.

As *link keys* podem ser subdivididas em quatro tipos distintos de chaves:

- Chaves de Combinação (*Combination Keys*): chave derivada a partir de informações de dois dispositivos que se comunicam;
- Chaves de Unidade (*Unit Keys*): chave específica para cada dispositivo Bluetooth gerada no momento da sua primeira utilização;
- Chave Mestre (*Master Key*): chave temporária que substitui uma *link key* durante uma determinada sessão. Pode ser utilizada quando o dispositivo mestre deseja transmitir informação para mais de um destinatário;
- Chave de Inicialização (*Initialization Key*): utilizada como *link key* durante o processo de inicialização de uma sessão, quando ainda não existem chaves de unidade ou de combinação.

Além das chaves acima, existe ainda um Número de Identificação Pessoal (*Personal Identification Number, PIN*) composto de 1 a 16 octetos, escolhido pelo usuário. O tamanho comumente utilizado é de 4 dígitos, aplicações que demandam por maior segurança podem utilizar PINs mais extensos. O PIN pode ser requisitado ao usuário do dispositivo durante o processo de conexão com a rede ou a ambos os usuários quando no processo de inicialização de comunicação ponto-a-ponto.

No processo de geração de qualquer uma das chaves citadas do tipo *link key*, com exceção da chave de encriptação, se utiliza o algoritmo E2 [42]. Este algoritmo possui dois modos: o E21, utilizado para a geração das chaves de unidade e de combinação; e o E22, utilizado para a geração das chaves de inicialização e mestre. O algoritmo E22 recebe como entrada os parâmetros RAND, PIN e tamanho do PIN.

O esquema de autenticação Bluetooth utiliza uma estratégia de desafio-resposta para descobrir se o outro dispositivo compartilha a mesma chave secreta. O protocolo utiliza o algoritmo E1, considerado como um algoritmo de autenticação computacionalmente seguro, baseado no algoritmo de encriptação SAFER+. Este, por sua vez, é uma versão melhorada do *block cipher* SAFER-SK128 de 64 bits [42].

A encriptação na camada de enlace somente pode ser ativada após o processo de autenticação do dispositivo. O algoritmo utilizado é o *stream cipher* E0. Ataques diretos ao E0 são conhecidos, mas são significantemente complexos. Jakobsson

and Wetzel em [43] apresentam dois ataques, o primeiro possui complexidade na ordem de 2^{100} e o segundo na ordem de 2^{66}.

O Bluetooth apresenta vulnerabilidades de segurança, tais como validação de endereço de dispositivos, estados inválidos e chaves expostas [44]. Além destas vulnerabilidades, as redes Bluetooth estão expostas a ataques, por exemplo, do tipo "*man-in-the-middle*", que permite o roubo da identificação do dispositivo e da chave de encriptação antes mesmo de iniciar uma sessão, podendo ser utilizadas para personificar e/ou recuperar informações de comunicações futura. Esta exposição não é específica do Bluetooth, os sistemas de troca de chaves em geral apresentam esta vulnerabilidade. Uma forma de minimizá-la é incluir suporte a sistemas de autenticação baseados em certificação digital.

Outra vulnerabilidade presente no Bluetooth está relacionada ao PIN [45]. Os dispositivos, em sua maioria, utilizam PINs extremamente pequenos (4 dígitos). Embora seja uma propriedade de implementação e não da especificação Bluetooth, o atacante pode realizar, através de busca exaustiva, a descoberta de chaves através destes PINs curtos. Por fim, uma vez que o endereço do dispositivo está associado a um usuário, um dispositivo intruso pode modificar seu endereço, passando a utilizar o mesmo do dispositivo do usuário e, assim, realizar a personificação.

2.5.2 Cenários de Utilização

A Figura 2.5 ilustra os cenários de utilização do Bluetooth no ambiente de computação móvel considerado neste trabalho, são eles:

- cenário 1: corresponde à Piconet 1 formada entre o DM1 e o SA;
- cenário 2: está representado através da Piconet 2, formada a partir da comunicação do DM2 com o PA 1 Bluetooth;
- cenário 3: compreende a Piconet 3, formada a partir da comunicação entre os DM2 e DM3.

No cenário 1, a segurança do enlace Bluetooth obecede aos requisitos de segurança fim-a-fim desejada às aplicações, pois apenas o enlace sem fio está compreendido entre os nós extremos da comunicação, ou seja, o DM e o SA. Portanto, os mecanismos de segurança apresentados na Seção 2.5.1 responsáveis pela segurança do enlace sem fio serão utilizados para a garantia da segurança fim-a-fim às aplicações. Conforme foi citado na Seção 2.5.1, no entanto, o Bluetooth apresenta vulnerabilidades

de segurança além de estar sujeito a ataques, tais como do tipo "*man-in-the-middle*". Sendo assim, a comunicação na Piconet 1 pode sofrer intervenção externa ocasionando a violação dos requisitos de segurança definidos na Seção 2.1. Além disso, este cenário é pouco comum no ambiente em questão, pois limita de forma considerável a mobilidade do DM1.

Analisando o cenário 2, observa-se que apenas o enlace sem fio compreendido entre o DM2 e o PA1 está encriptado (utilizando o algoritmo E0). Sendo assim, os dados trafegam em aberto no trecho de rede compreendido do PA1 até o SA. Portanto, a segurança fim-a-fim não é oferecida, permitindo a recuperação das informações transmitidas pelo DM ou para este, através de escutas promíscuas do meio, por usuários da rede local ou oriundos da Internet.

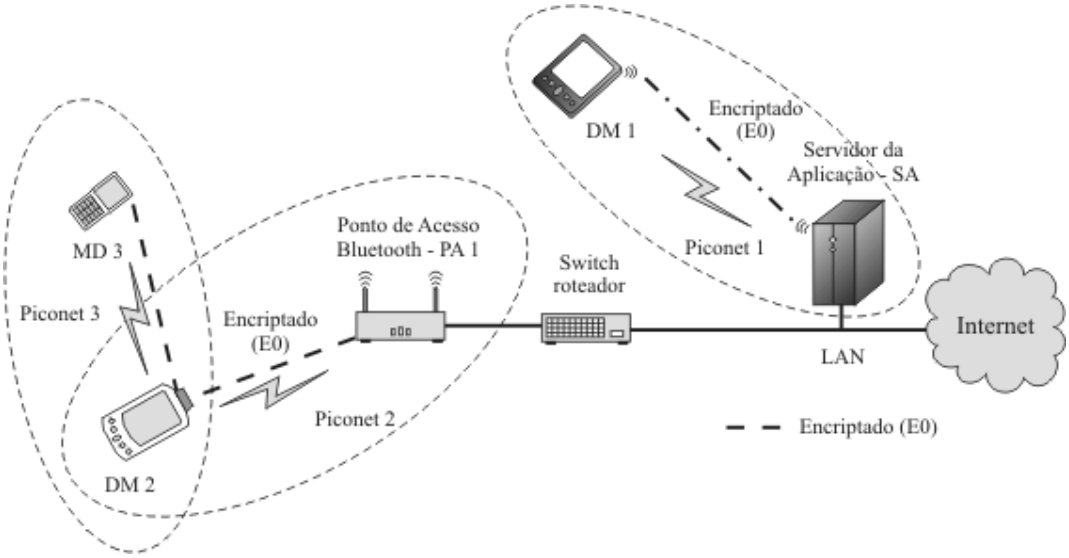

Figura 2.5. Comunicação entre os DMs e o SA utilizando o sistema Bluetooth.

Por sua vez, no cenário 3, a comunicação ponto-a-ponto entre os DM2 e DM3 está protegida através da encriptação do enlace sem fio. Entretando, este cenário não apresenta troca de informações com o SA, que é o extremo final para o qual os dados das aplicações se destinam. Caso o DM2 opere com ponte entre as Piconets 2 e 3 de forma a permitir a comunicação do DM3 com o SA, os mesmo problemas de segurança descritos do cenário 2 irão ocorrer.

Por fim, conclui-se que existem falhas na provisão de segurança fim-a-fim para as aplicações no ambiente de computação, ao se utilizar o Bluetooth com sistema de comunicação sem fio.

2.6 Conclusão

Neste capítulo foram apresentados o ambiente de computação móvel considerado neste trabalho, bem como os sistemas de comunicação sem fio e seus aspectos de segurança.

Apesar da existência de mecanismos de segurança inerentes a cada sistema, estes não obedecem aos requisitos de segurança fim-a-fim exigidos pelas aplicações no ambiente de computação móvel considerado aqui, oferecendo apenas, na maioria dos sistemas, a segurança do enlace sem fio compreendido entre os dispositivos móveis e a rede de acesso. No capítulo seguinte, serão discutidas soluções existentes para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel, objetivo principal deste trabalho.

3 Soluções de Segurança fim-a-fim para Aplicações em Computação Móvel

Este capítulo apresenta soluções para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel considerado neste trabalho. Serão discutidas as vantagens e desvantagens de cada solução, bem como a justificativa para o desenvolvimento da proposta do Framesec e da ferramenta PEARL.

3.1 Introdução

No Capítulo 2 foram apresentados os sistema de comunicação sem fio, bem como foram discutidos os aspectos de segurança dos mecanismos inerentes a cada sistema. Os sistemas, em sua maioria, oferecem apenas proteção dos dados no enlace sem fio compreendido entre os DMs e a rede de acesso ao sistema, como pode ser visto na Figura 2.1. Além de não prover a segurança fim-a-fim desejada às aplicações, do DM ao SA, estes sistemas apresentam vulnerabilidades que põem em risco os requisitos de segurança definidos na Seção 2.1. Portanto, os mecanismos de segurança oferecidos por esses sistemas são insuficientes para a provisão de segurança fim-a-fim desejada às aplicações no ambiente de computação móvel considerado neste trabalho.

Neste capítulo serão apresentados os protocolos de segurança que podem ser utilizados para a provisão de segurança fim-a-fim no ambiente de computação móvel, bem como soluções que utilizam ou não estes protocolos existentes.

3.2 Protocolos de Segurança

A segurança fim-a-fim desejada às aplicações no ambiente de computação móvel considerado neste trabalho pode ser fornecida através da utilização de protocolos de segurança, tais como o SSL/TLS. A utilização destes protocolos é condicionada ao sistema de comunicação sem fio utilizado pelo DM para se comunicar com o SA, ou seja, de acordo com o sistema de comunicação sem fio utilizado, os protocolos que poderão ser utilizados pela aplicação para a provisão da segurança fim-a-fim serão diferentes.

Nesta seção serão discutidos os mecanismos de segurança inerentes aos protocolos SSL/TLS, o S/MIME e o WTLS. O protocolo IPSec não será tratado nesta seção visto que o mesmo já foi descrito na Seção 2.2.

3.2.1 SSL/TLS

O SSL/TLS (*Secure Socket Layer/Transport Layer Security*) [2] é um protocolo padrão especificado pelo IETF, desenvolvido com a finalidade de oferecer serviços de segurança a uma vasta classe de aplicações de redes interativas, tais como de comércio eletrônico. O SSL está disponível nas versões 1.0, 2.0 e 3.0. O TLS, por sua vez, está na primeira versão e corresponde à versão 3.0 do SSL.

O SSL/TLS opera no topo do TCP e fornecendo aos protocolos (e.g., HTTP) ou aplicações, serviços de confidencialidade e integridade dos dados transmitidos entre as duas partes comunicantes, bem como autenticação mútua através da utilização de certificados digitais. No caso em questão, as duas partes comunicantes referem-se ao DM e o SA.

O protocolo TLS, que é corresponde à versão 3.0 do SSL conforme citado anteriormente, é composto por duas camadas: o *TLS Record Protocol* e o *TLS Handshake*. No nível mais baixo do protocolo, que realiza a interface com o TCP, está o *TLS Record Protocol*.

O *TLS Record Protocol* fornece a segurança fim-a-fim através das seguintes propriedades:

- a conexão é privada. O protocolo utilizando algoritmos de criptografia simétrica (veja a Seção B.1) para a encriptação dos dados. As chaves utilizadas são geradas unicamente para cada conexão e são baseadas em negociação prévia realizada pelo protocolo *TLS Hashskake*. O *TLS Record Protocol* pode também ser utilizado sem encriptação;
- A conexão é confiável. O transporte das mensagens inclui uma verificação da integridade da mensagem usando as funções *hash* chaveadas MAC (veja a Seção B.3). O *TLS Record Protocol* pode operar sem a utilização do MAC, no entanto isto só ocorre quando outro protocolo utiliza o *TLS Record Protocol* para transportar parâmetros de negociação de segurança.

O *TLS Record Protocol* é utilizado para encapsular os protocolos de um nível mais elevado na pilha de protocolos. Um dos protocolos encapsulados é o *TLS*

Handshake, que permite a negociação prévia dos algoritmos e das chaves de criptografia utilizados entre as partes comunicantes. O *TLS Handshake Protocol* provê conexão segura através das três propriedades básicas:

- os pontos comunicantes são autenticados utilizando criptografia assimétrica (veja a Seção B.2). Esta autenticação pode ser opcional, mas geralmente é utilizada por uma das partes comunicantes (e.g., o SA realiza a autenticação do DM);
- a negociação da chave secreta é segura, não permitindo a sua recuperação por atacantes que não participam da conexão;
- a negociação é confiável. Nenhum atacante pode interferir na comunicação da negociação sem ser detectado pelas partes comunicantes.

Entretanto, a utilização do SSL/TLS requer poder de processamento e memória nem sempre presentes nos DMs, já que grande parte das operações realizadas por este protocolo é baseada na criptografia de chave pública e em certificados digitais.

Buscando uma alternativa que substitua o SSL/TLS de menor peso computacional, a Sun ® [71] desenvolveu o KSSL (*kilobyte SSL*) [3], que é uma implementação somente do lado cliente (*client-side-only*) do SSL 3.0 (TLS 1.0). O KSSL, assim como o SSL/TLS, assume a presença do TCP [6]. Desta forma, a sua utilização é impossibilitada nas aplicações para DMs que não apresentam suporte à pilha de protocolos TCP/IP.

3.2.2 S/MIME

O *Secure/Multipurpose Internet Mail Extensions* (S/MIME) [9] é um protocolo padrão utilizado para prover serviços seguros orientado a mensagens, tais como o envio de correio eletrônico. Baseado no padrão popular Internet MIME, o S/MIME prover serviços seguros para aplicações de mensagens eletrônicas, garantindo a privacidade e a integridade das mensagens através da utilização da criptografia, e a autenticidade e o não-repudio da origem através do uso de assinaturas digital. O núcleo das operações de segurança do S/MIME utiliza a criptografia de chave pública bem como o uso em larga escala de certificação digital.

O S/MIME foi originalmente desenvolvido pela *RSA Data Security, Inc* [17]. O formato da mensagem utilizado no S/MIME é baseado no PKCS #7 [47] e os certificado no formato X.509v3 [47].

O S/MIME pode ser utilizado como solução de segurança em aplicações que utilizam a arquitetura de distribuição *store-and-forward* (armazena e depois entrega), tais como de correio eletrônico. Entretanto, no escopo do ambiente de computação móvel considerado neste trabalho, a utilização deste protocolo limitaria a aplicabilidade dos DMs, já que as aplicações em tempo real ou *on-line* exigem a transmissão imediata de dados e o S/MIME foi desenvolvido para ser utilizado por aplicações que utilizam a arquitetura de distribuição *store-and-forward,* que não é o caso em questão.

Outro problema existente na utilização do S/MIME no ambiente de computação considerados neste trabalho, está relacionado ao fato de que este protocolo utiliza exaustivamente a criptografia de chave pública para o provimento dos serviços de autenticação do emissor e não-repúdio de informações transmitidas. Portanto, em DM que apresentam limitações de recursos, tais como o Palm M130, a utilização deste protocolo denigre consideravelmente o desempenho da aplicação de forma a torná-la inviável do ponto de vista operacional.

3.2.3 WTLS

Tendo em vista aos problemas apresentados na utilização dos protocolos SSL/TLS e S/MIME discutidos nas seções anteriores, poder-se-ia ainda utilizar o WTLS (*Wireless Transport Layer Security*) [11], que é um protocolo integrante da pilha de protocolos WAP (*Wireless Application Protocol*) [13], para garantir a segurança das informações transmitidas por aplicações que utilizam o protocolo HTTP para realizar suas interações com a rede.

O WTLS provê autenticação do emissor, confidencialidade e integridade dos dados. Este protocolo é baseado no SSL/TLS, apresentado na Seção 3.2.1, desenvolvido pelo IETF. O WTLS é utilizado para provê a comunicação segura entre um DM WAP e o gateway WAP. Existem três diferentes classes do WTLS:

- Classe 1: Este tipo implementa o mecanismo de troca de chaves não autenticadas Diffie-Hellman para estabelecer a chave de sessão;
- Classe 2: esta classe força a autenticação do lado do servidor utilizando chaves públicas e certificação digital, similar ao utilizado no protocolo SSL/TLS. O gateway WAP utiliza o certificado WTLS, que possui formato de certificado reduzido e é baseado no certificado X.509 [47].

– Classe 3: clientes que implementam este nível são capazes de utilizar a autenticação do lado cliente utilizando certificados. Estes certificados podem ser armazenados no cliente (DM) ou no servidor publicamente acessível.

Os DM WAP que implementam o WTLS da Classe 1, oferecem mecanismos insuficientes para a provisão de segurança fim-a-fim desejada às aplicações no ambiente de computação móvel considerado neste trabalho, diferentemente das Classes 2 e 3, conforme os requisitos definidos na Seção 2.1.

Entretanto, a utilização do WTLS apresenta vulnerabilidade. Conforme citado anteriormente, o WTLS é utilizado para provê a segurança do DM WAP até o *gateway* WAP. Portanto, a segurança é fim-a-fim do DM até o *gateway* WAP, e não até o SA. Desta forma, afim de tornar a segurança fim-a-fim até o SA deve ser utilizado o SSL/TLS para estabelecer uma comunicação segura entre o *gateway* WAP e o SA. A vulnerabilidade de segurança existe devido ao mecanismo de conversão de protocolo de segurança realizado no *gateway* WAP [12], que coverte os protocolos WTLS em SSL e vice-versa.

Esta conversão de protocolo mantém no *gateway* WAP os dados transmitidos pela aplicação, em determinado momento, sem nenhuma criptografia (i.e., os dados estão em texto claro), acarretando em riscos contra a confidencialidade e integridade das informações no *gateway* WAP.

Com o aprimoramento do WAP, que ocorreu na definição da especificação do WAP 2.0 [13], houve a eliminação desta vulnerabilidade por não mais existir o *gateway* WAP entre a comunicação entre os DMs e o SA. A partir desta especificação, foi adicionado suporte a todos os protocolos padrões de comunicação Internet, tais como o IP, TCP e HTTP, tornando assim o DM visível à Internet. A atualização para o WAP 2.0, entretanto, requer atualização dos DM (i.e., a troca do aparelho) e da rede sem fio. Portanto, esta nem sempre será uma solução viável.

Na próxima seção serão apresentadas soluções de segurança fim-a-fim propostas para o ambiente de computação móvel considerado nestra trabalho.

3.3 Soluções de segurança fim-a-fim

Em [6] é apresentada uma solução, desenvolvida na camada de aplicação, para a provisão de segurança fim-a-fim no ambiente de computação móvel. A solução proposta

é desenvolvida na plataforma J2ME e provê a autenticação e a confidencialidade dos dados transmitidos entre um DM cliente, que possui suporte à plataforma J2ME, e um servidor de aplicação, baseado na plataforma J2EE. O objetivo dessa solução é prover um ambiente altamente seguro de simples utilização e que não requer qualquer modificação na infraestrutura de comunicação ou protocolos utilizados nas redes sem fio.

Essa solução utiliza o algoritmo de criptografia simétrica AES Rijndael (veja a Seção B.1.3) com chaves e blocos de 128 bits para realizar a encriptação dos dados, garantindo a confidencialidade das informações transmitidas.

A autenticação é fornecida através do mecanismo de troca de chaves utilizado na solução. Este mecanismo utiliza duas chaves de sessão geradas randomicamente a cada nova sessão iniciada, sendo que uma das chaves é utilizada para encriptar as informações no cliente (DM) e decripta-las no servidor e a outra para encripta-las no servidor e decripta-las no cliente. O mecanismo de troca de chaves de sessão funciona da seguinte forma: o SA, ao receber uma requisição da aplicação cliente no DM, gera randomicamente um par de chaves que será armazenado na entrada específica do DM no repositório de chaves seguro, localizado no SA ou em outro servidor confiável. O SA encripta esse par de chaves utilizando um *pincode* (código de acesso da aplicação cliente) de 64 bits concatenado com uma chave secreta de 64 bits conhecida somente pelo servidor e o cliente. Este par de chaves de sesssão é enviado ao DM onde será armazenado em variáveis locais da aplicação. A autenticação do emissor é realizada, portanto, utilizando a chave secreta conhecida apenas pelo SA e o DM.

Entretando, esta solução não fornece as aplicações mecanismos de segurança para o provimento da integridade, bem como para o não-repúdio das informações transmitidas. Além disso, o algoritmo utilizado AES Rijndael pode não ser suportado ou possuir baixo desempenho de execução em DM que possuem limitações de recursos, sendo necessário a substituição deste por outro algoritmo de criptografia simétrica que apresente melhor desempenho no DM utilizado.

Em [77], os autores apresentam uma solução para o desenvolvimento de aplicações seguras para dispositivos móveis, chamado MobiS (Mobilidade e Segurança). O MobiS possui 3 (três) versões de segurança: o MobiS Simétrico, o MobiS Assimétrico e o MobiS Híbrido. As denominações das versões fazem referência ao tipo de algoritmo de criptográfico utilizado. Sendo assim, o MobiS Simétrico utiliza algoritmos de criptografia simétrica, o MobiS Assimétrico utiliza algoritmos de

criptografia assimétrica e, por último, o MobiS Híbrido, que combina a utilização dos dois tipos de algoritmos.

No trabalho apresentado, os autores descreveram a versão MobiS Simétrico, que é uma solução baseada no protocolo WEP [14], utilizado em redes IEEE 802.11b para prover a confidencialidade dos dados, conforme citado na Seção 2.2. A solução MobiS Simétrico está Figura 3.1. Observando a figura, em I são mostradas as operações de segurança realizadas antes do envio da mensagem. Por sua vez, em II são mostradas as operações realizadas ao recebê-las.

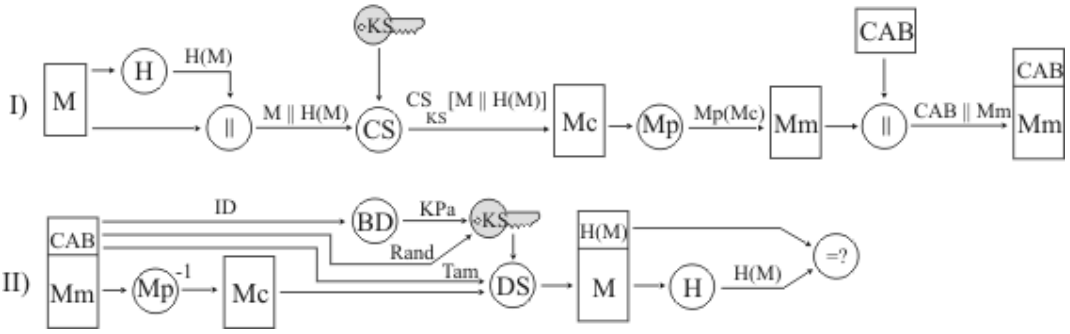

Figura 3.1. Versão MobiS Simétrico (baseado em [77])

O processo I da seguinte forma: a partir da mensagem M a ser enviada, é calculado um resumo, H(M), utilizando a função *hash* H. O resumo é então concatenado à mensagem gerando o texto resultante a ser cifrado, ou seja, M || H(M). O texto resultante é cifrado utilizando um algoritmo de criptografia simétrico *block cipher* CS (e.g., *Triple* DES) utilizando a chave de sessão KS, assim, $Mc = CS_{KS} [M \| H(M)]$. A chave de sessão pode ser gerada das seguintes formas: concatenando a chave simétrica KPa presente no dispositivo com um *Rand* calculado a cada transmissão (KS = KPa || Rand) ou executando uma operação Xor (ou exclusivo) entre KPa e o *Rand* gerado (KS = KPa ⊕ Rand).

Após cifrar a mensagem, é aplicado sobre o Mc um algoritmo de conversão Mp (e.g., Base64) transformando caracteres inválidos gerados na criptografia em caracteres aceitáveis para a transmissão (Mm = Mp(Mc)). Este processo de mapeamento deve ser realizado, pois os algoritmos de criptografia geram caracteres que não são transmitidos corretamente por alguns protocolos, tais com o S/MIME e o HTTP. Ao Mm é adicionado um cabeçalho, CAB, contendo os campos: tamanho da mensagem original (TAM), o identificador do dispositivo (ID), o identificador de combinações de algoritmos (ICA) (e.g., um identificador para a combinação *Triple* DES + MD5 + *HexEncoder*) e o *Rand* gerado.

O envio do campo TAM no CAB é necessário, pois os algoritmos de criptografia simétrica *block cipher* requer que o tamanho da entrada seja múltiplo do tamanho do bloco que o algoritmo trabalha (e.g., o Rijndael trabalha com blocos de 16 bytes). Sendo assim, é realizado antes da criptografia um *padding* ao final da mensagem para tornar seu tamanho múltiplo do tamanho do bloco do algoritmo.

Por sua vez, o processo de recebimento da mensagem ocorre conforme a seguir: ao receber a mensagem, o CAB é lido e o processo inverso de mapeamento (Mp^{-1}) é realizado sobre o restante da mensagem. Através do ID, é possível recuperar a chave simétrica KPa no banco de dados de chaves (BD). A chave KPa é concatenada (ou realizado um Xor como explicado anteriormente) com o *Rand* enviado, gerando a chave KS que será utilizada para decifrar a mensagem (DS). A função *hash* é aplicada sobre os dados decifrados, gerando um resumo que será comparado ao resumo que foi enviado. Se forem iguais, a integridade da mensagem foi assegurada.

A robustez do MobiS Simétrico está diretamente relacionada à combinação dos algoritmos bem como ao tamanho da chave de criptografia.

Um dos problemas presentes na criptografia simétrica é a vulnerabilidade a ataques de texto cifrado conhecido (*known cipher text*). No MobiS Simétrico este problema é resolvido através do uso do número randômico *Rand* na composição da chave de sessão KS.

O MobiS Simétrico garante a confidencialidade dos dados, pois somente as partes comunicantes (dispositivo móvel e servidor) podem recuperar os dados originais. A integridade dos dados é assegurada através da função *hash*. Caso os dados sejam alterados, o resumo calculado no recebimento não coincidirá com o resumo enviado.

A fabricação de dados é evitada através da criptografia simétrica, pois somente quem possui a chave poderá produzir uma mensagem válida. Ataques de *reply* podem ser realizados, o que deve ser resolvido pela aplicação. Uma possível solução seria adicionar um código incremental que identifica unicamente os dados enviados. Assim, o servidor poderá descartar os dados recebidos com o mesmo código.

O MobiS Simétrico não fornece as aplicações os serviços de segurança de autenticação do emissor e o não-repúdio das informações. Além disso, não faz qualquer consideração quanto ao desempenho dos algoritmos de criptografia utilizados na solução, apenas apresenta um estudo de caso que utiliza os algoritmos AES Rijndael e o MD5.

Estas foram as soluções de segurança fim-a-fim encontradas na literatura, aplicáveis ao ambiente de computação móvel considerado neste trabalho. As soluções aprensentadas não fornecem suporte a todos os serviços de segurança fim-a-fim desejada as aplicações, conforme os requisitos definidos na Seção 2.1. Além disso, elas não mencionam qualquer consideração quanto ao desempenho dos algoritmos de criptografia utilizados.

3.4 Conclusão

Conforme apresentado neste capítulo, as soluções de segurança fim-a-fim existentes são insuficientes e até ineficientes para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel considerado neste trabalho. Portanto, se faz necessário adicionar mecanismos às aplicações para o provimento de segurança fim-a-fim.

No entanto, a construção desses mecanismos não é uma tarefa fácil de ser realizada por desenvolvedores de aplicações leigos em segurança da informação. Portanto, o Framesec desenvolvido e proposto neste trabalho tem por finalidade fornecer a especificação de blocos semifinalizados com propósitos de segurança, que podem ser instanciados e especializados para a construção de mecanismos de segurança a serem incorporados às aplicações que possuem requisitos de segurança no ambiente de computação móvel.

O mecanismo de segurança construído a partir do Framesec utiliza os algoritmos de criptografia de acordo com o serviço de segurança definido nos requisitos da aplicação. O desenvolvedor de aplicações necessita de parâmetros de desempenho dos algoritmos criptográficos para realizar a escolha dos algoritmos que serão incorporados aos mecanismos de segurança construídos a partir do Framesec.

Portanto, este trabalho propõe também a ferramenta PEARL, que permite ao desenvolvedor avaliar o desempenho dos algoritmos criptográficos que irão compor os mecanismos de segurança construídos a partir do Framesec no DM no qual será utilizado.

4 Framesec: um Framework para Provisão de Segurança fim-a-fim para Aplicações no Ambiente de Computação Móvel

O presente capítulo apresenta o *framework* para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel. Antes de descrevê-lo em detalhes, serão discutidos os conceitos relacionados a *frameworks*, as metodologias para sua construção e padrões de projeto relacionados.

4.1 Introdução

Segundo Viljamaa [90], *Framework* é um conjunto de objetos reutilizáveis que engloba conhecimento de determinadas áreas e se aplica a um domínio específico. Uma aplicação completa ou parte significante dela pode ser especializada a partir dessa estrutura fazendo-se as adaptações necessárias ou adicionando-se novas características ao *framework*. Um *framework* determina então a arquitetura da aplicação e predefine parâmetros de projeto, permitindo ao projetista ou desenvolvedor concentrar-se nos detalhes específicos da aplicação.

Framework é definido também como um projeto de reutilização que descreve como o sistema está decomposto em um conjunto de objetos que interagem entre si, podendo ser uma aplicação inteira ou apenas um subsistema. Jonhson e Foote [91] definem que *frameworks* descrevem a interface de cada objeto e o fluxo de controle entre eles.

Em [93], Gamma define *frameworks* como um conjunto de classes que tornam um projeto reutilizável para uma classe específica de software. O desenvolvedor o adapta para uma aplicação particular através de subclasses e composição de instâncias das classes do *framework*.

Por sua vez, Taligent [94] classifica os *frameworks* de acordo com os domínios de problemas que eles abrangem, conforme a seguir:

- *frameworks de suporte:* provê serviços no nível de sistemas, tais como acesso a arquivos, suporte a computação distribuída ou a dispositivos de *drivers*. Desenvolvedores de aplicações tipicamente utilizam *frameworks* de suporte diretamente ou modificações produzidas pelos provedores de

sistema. Contudo, *frameworks* de suporte podem ser especializados – por exemplo, no desenvolvimento de um novo sistema de arquivos ou dispositivo controlador;

– *frameworks de aplicações:* encapsulam conhecimento aplicável a uma variedade de programas, por exemplo, interface de usuário gráfica (*Graphic User Interface, GUI*) para aplicações comerciais;
– *frameworks de domínio:* encapsulam o conhecimento de um domínio de problema particular, por exemplo, *framework* de segurança.

Um *framework* tem uma representação física em termos de classes e métodos, onde não apenas a implementação é reutilizável, mas também a sua estrutura. As partes do *framework* que são abertas a extensões e especializações são denominadas de *hot spots* [90] – áreas variáveis ou pontos de flexibilidade. Os *hot spots* tornam os *frameworks* flexíveis, são como lacunas no domínio da aplicação, sendo tarefa do desenvolvedor da aplicação preenchê-las com uma solução própria ou de terceiros [85].

Os *frameworks* são ainda classificados quanto ao suporte ao usuário, podendo ser do tipo caixa branca (o usuário constrói as classes a partir das classes disponíveis), caixa preta (o usuário tem que escolher uma das classes fornecidas) ou caixa cinza (permite a construção de classes ou a utilização de classes fornecidas).

Segundo Lajoie e Keller [95], os *frameworks* não são bibliotecas de classes. Um *framework* é o projeto de um conjunto de classes que colaboram para realizar um conjunto de responsabilidades. Enquanto os componentes das bibliotecas de classes são utilizados individualmente, as classes no *framework* são reutilizadas como um todo para resolver uma instância específica de certo problema. Portanto, a reutilização promovida pela abordagem de *frameworks* se situa num patamar de granularidade superior à reutilização de classes, por reusar classes interligadas ao invés de classes isoladas. Da perspectiva do desenvolvedor de aplicações, a maior diferença entre um *framework* e uma biblioteca de classes está no conhecimento necessário para utilizá-los. Os usuários da biblioteca de classes precisam somente entender a interface externa das classes e definir toda a estrutura de sua aplicação. Entretanto, usuários de *frameworks* devem entender o projeto abstrato do mesmo bem como a estrutura de suas classes, de forma a adaptá-las ou estendê-las.

A complexidade em se desenvolver *frameworks* deve-se aos seguintes fatores [89]:

– necessidade de considerar os requisitos de um conjunto significativo de aplicações de modo a torná-lo genérico;
– necessidade de ciclos de evolução voltados a dotar a estrutura de classes do *framework* de capacidade de alterar suas funcionalidades sem conseqüências imprevistas sobre o conjunto da estrutura e extensibilidade (capacidade de ampliar a funcionalidade presente sem conseqüências imprevistas sobre o conjunto da estrutura).

Em termos práticos, construir um *framework* genérico, extensível e com capacidade de sofrer alterações, requer uma cuidadosa identificação das partes que devem ser flexíveis e a seleção de soluções de projetos de modo a produzir uma arquitetura bem estruturada. Isto conduz a observação de princípios de projeto orientados a objetos.

Em relação ao projeto de um *framework*, as características são oferecidas em dois níveis: características do domínio, que são características relevantes do domínio úteis às aplicações; e características estruturais, que são características que facilitam a adaptação e a evolução do *framework*. As características estruturais, por sua vez, incluem um arranjo de lógica (projeto) e estrutura física (classes) do *framework*.

Existem estudos e metodologias para o desenvolvimento de *frameworks*. Entre as metodologias existentes encontram-se as de Johnson (1993), Pree (1999), Taligent (1994) e Carneiro (2003).

Em [96], Johnson apresenta um processo chamado de "Projeto Dirigido por Exemplos", composto de etapas de análise, projeto e teste. Ele afirma que as pessoas pensam de forma concreta e não de forma abstrata, assim a abstração de domínio é obtida a partir da generalização de casos concretos – as aplicações.

As abstrações são obtidas de uma forma *bottom up*, a partir do exame de exemplos concretos onde os aspectos semelhantes de diferentes aplicações podem dar origem a classes abstratas que agrupam as semelhanças, cabendo às classes concretas do nível hierárquico inferior fornecerem a especialização para satisfazer cada caso. A forma tida como ideal para desenvolver um *framework*, segundo Johnson, é:

– analisar o domínio do problema - assimilar as abstrações já conhecidas, coletar exemplos de programas que podem ser construídos a partir do *framework* (mínimo de quatro ou cinco);
– projetar abstrações que podem ser especializadas para abranger os exemplos;

– testar o *framework* utilizando-o para desenvolver os exemplos. Cada exemplo é uma aplicação separada e a execução do teste consiste em implementar o software a partir do *framework.*

A metodologia proposta por Taligent [94] consiste no desenvolvimento de *frameworks* estruturalmente menores e mais simples, que utilizados conjuntamente darão origem às aplicações. Sua justificativa é que pequenos *frameworks* são mais flexíveis e podem ser reutilizados com maior freqüência. Assim, a ênfase passa a ser o desenvolvimento de *frameworks* pequenos e direcionados a aspectos específicos do domínio. A metodologia propõe a seqüência de quatro passos:

– analisar o domínio do problema: consiste em analisar o domínio e identificar os *frameworks* necessários para o desenvolvimento de aplicações, examinar soluções existentes e validar as informações com especialistas do domínio;
– definição da arquitetura e projeto: consiste no refinamento da estrutura de classes obtida no passo anterior, aperfeiçoamento do projeto com o uso de padrões de projeto e validar as informações com especialistas do domínio;
– implementação do *framework*: implementar as classes principais, testar o framework (gerando aplicações) e iterar para refinar o projeto;
– disponibilizar o *framework* com documentação e suporte técnico ao cliente.

Pree [86] propõe a metodologia de construção de "Projeto Dirigido por *Hot Spot*", onde a essência é identificar os *hot spots* na estrutura de classes de um domínio e, a partir disso, construir o *framework.* As etapas são as seguintes:

– na primeira etapa são realizadas a identificação de classes com a ajuda de especialistas de domínio;
– na segunda etapa são identificados os *hot spots*, que devem ser documentados com um tipo de documento chamado de cartão de *hot spots*;
– a terceira etapa, o projeto do *framework*, consiste em modificar a estrutura de classes inicialmente definida, de modo a comportar a flexibilidade requerida. Essa etapa é centrada no uso de padrões de projeto para garantir a flexibilidade;
– a quarta etapa consiste no refinamento da estrutura do *framework* a partir de novas intervenções de especialistas do domínio. Se após isto o *framework* for avaliado como satisfatório, está concluída uma versão do *framework*, senão, retorna-se à etapa de identificação de *hot spots* (i.e., segunda etapa).

Por fim, Carneiro [88] propõe uma abordagem iterativa e incremental. O *framework* é refinado e ajustado ou são adicionados novos requisitos através dos vários ciclos de desenvolvimento iterativo. Cada ciclo trata de um conjunto relativamente pequeno de requisitos. É incremental, pois adiciona novas funcionalidades ao *framework* durante os vários ciclos de desenvolvimento. O *framework* pode ser composto de múltiplos ciclos de desenvolvimento iterativo e a cada ciclo ele contém mais funcionalidades que o anteriormente gerado.

Os *frameworks* mantêm um estreito relacionamento com padrões de projeto e são freqüentemente entendidos como sendo padrões de larga escala. O relacionamento entre padrões de projeto e *frameworks* é importante, uma vez que padrões tipicamente têm o foco nos aspectos de flexibilidade de software e flexibilidade é exatamente o que é necessário em *frameworks* para possibilitar sua especialização nas aplicações reais.

Portanto, os padrões de projeto podem ser utilizados como uma base para construção de *frameworks* de aplicação. Atualmente é amplamente aceito que padrões de projeto são bem adequados para documentar *frameworks*. Padrões também permitem o entendimento do *framework*, sem forçar o conhecimento pleno do seu código fonte [90].

4.2 Metodologia de Desenvolvimento Utilizada na Construção do Framesec

Dentre as várias definições de *framework* descritas na Seção 4.1, a considerada neste trabalho é a realizada por Viljamaa [90], por ser a que melhor traduz conceitualmente a proposta do Framesec, que consiste de um *framework* de aplicação que engloba o conhecimento do domínio específico de segurança.

A metodologia de desenvolvimento de *frameworks* utilizada para a construção desta proposta foi a abordagem iterativa e incremental definida por Carneiro [88]. A característica determinante para a realização desta escolha é que esta metodologia mantém controlada a complexidade de construção do *framework*, nos vários ciclos iterativos e incrementais de seu desenvolvimento, ao contrário das outras metodologias. Utilizando esta metodologia, obtêm-se as vantagens de que a complexidade não se torna incontrolável, a realimentação é gerada mais cedo no processo (a cada novo ciclo) e a documentação de cada fase serve para o entendimento do *framework* e a sua futura utilização no desenvolvimento de aplicações.

Conforme mencionado na seção anterior, a abordagem de desenvolvimento de *frameworks* iterativa e incremental é composta dos seguintes passos: o primeiro corresponde à análise de domínio, passando para a fase de construção do *framework*. Em seguida, são realizados vários ciclos de desenvolvimento e, a cada ciclo, os requisitos são adicionados ou refinados, é então realizada uma análise, os *hot spots* são avaliados e o projeto é alterado para refletir os novos requisitos ou refinamentos.

A cada ciclo, um *framework* é implementado e é criada uma instância exemplo. Deve-se elaborar um roteiro de utilização ao final de cada ciclo para facilitar o entendimento e a instanciação de aplicações a serem geradas a partir do *framework* construído. A Figura 2.1 ilustra os ciclos desta metodologia.

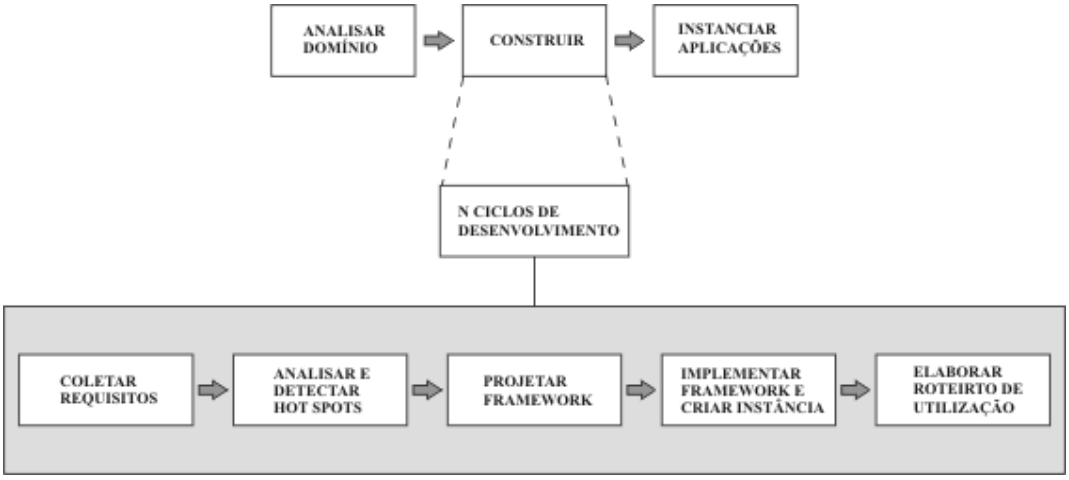

Figura 4.1. Metodologia iterativa e incremental (figura baseada em [88]).

A fase de análise de domínio fornecerá a entrada para o desenvolvimento do *framework*. Trata-se de um processo através do qual as informações a serem utilizadas no desenvolvimento do *framework* serão identificadas, capturadas e organizadas. Pode ser vista como uma análise ampla que tenta obter os requisitos do domínio do problema, incluindo requisitos futuros. Dois documentos importantes são gerados como resultado da análise de domínio: o escopo do domínio e um modelo estático contendo objetos importantes do domínio.

O próximo passo, correspondente à construção do *framework,* é composto pelas seguintes etapas:

– Coletar requisitos: os requisitos são descrições das necessidades ou dos desejos para um produto. O objetivo básico da fase de requisitos é identificar e documentar o que realmente é necessário, em uma forma que

comunica claramente essa informação ao cliente e aos membros da equipe de desenvolvimento [92];

- Analisar e detectar *hot spots*: a detecção de *hot spots* será efetuada verificando-se no modelo de domínio, modelo de requisitos ou nas aplicações já existentes, que aspectos fixos podem ser flexibilizados nas aplicações a serem desenvolvidas a partir do *framework* proposto. Para a construção do catálogo de *hot spots* do *framework*, a cada *hot spot* detectado devem ser preenchidas as informações para a seguir: número, nome do *hot spot*, descrição e possíveis exemplos de utilizações do aspecto variável;
- Projetar o *framework*: nessa fase, o modelo estático de objetos criado na fase de análise deve ser detalhado e os *hot spots* incluídos para gerar o diagrama de classe de projeto. Tanto os pontos de flexibilidade descritos no catálogo de *hot spots* são modelados quanto às partes fixas;
- Implementar *frameworks* e criar instâncias: com a finalização do diagrama de classes de projeto para o ciclo corrente de desenvolvimento há detalhes suficientes para gerar o código do *framework*. Um volume significativo de decisões e de trabalho foi realizado nas etapas anteriores, mas a fase de programação não é um passo trivial de geração de código. Segundo Larman [92], durante a codificação do *framework* serão realizadas muitas mudanças, bem como serão descobertos e resolvidos problemas;
- Elaborar roteiro de utilização: a elaboração de um roteiro ajudará o desenvolvedor a criar aplicações a partir do *framework*. Assim, o roteiro deverá conter informações suficientes para permitir o entendimento do framework e sua forma de trabalho.

Na fase de análise e detecção de *hot spots*, será construído o catálogo de *hot spots* do *framework*, conforme citado anteriormente. Nesta fase, a cada *hot spot* detectado serão definidos os campos número, nome do *hot spot*, descrição, o tipo de adaptação requerida bem como o grau de apoio oferecido pelo *hot spot*. A adaptação requerida pelo hot spot, segundo o Froehlich [87], podem ser um dos seguintes tipos dispostos na Tabela 4.1.

Os dois tipos mais comuns são: habilitar uma característica e adicionar uma característica. Habilitar uma característica consiste em ativar características que fazem parte do *framework,* mas podem não fazer parte da sua implementação padrão. Ao contrário de habilitar uma característica onde o desenvolvedor está utilizando serviços

existentes no *framework*, adicionar uma característica envolve adicionar algo que o *framework* não estava capacitado antes. As adições são freqüentemente feitas através da extensão de classes existentes ou adição de novas classes.

Tabela 4.1. Tipos de adaptação de *Hot Spot* (adaptado de [88]).

Tipo	Descrição
1	Habilitar uma característica
2	Desabilitar uma característica
3	Substituir uma característica
4	Aumentar uma característica
5	Adicionar uma característica

Desabilitar uma característica consiste em desativar uma característica que faz parte da implementação padrão do *framework*. Substituir uma característica consiste em desabilitar uma característica e colocar uma nova em seu lugar. Por fim, aumentar uma característica consiste em estender uma característica sem alterar o fluxo normal de controle estabelecida pelo *framework*. Essa alteração pode interceptar o fluxo de controle existente, executar alguma ação necessária e retornar o controle de volta ao *framework*.

Em relação ao grau de apoio fornecido pelo *framework* ao desenvolvedor, existem três tipos:

- *Opcional:* o desenvolvedor seleciona um componente pré-definido disponível numa biblioteca;
- *Suporte Padrão:* as adaptações estão associadas a instanciações pré-definidas que dependem de informações fornecidas pelo usuário da aplicação;
- *Ilimitado:* as adaptações são realizadas sem apoio do *framework*. O desenvolvedor tem liberdade para alterar o *framework* adicionando características não previstas na sua definição original.

Ao final de cada ciclo de desenvolvimento, um novo *framework* é criado contendo mais funcionalidades que o gerado no ciclo anterior, conforme os requisitos deste ciclo. Após a execução dos N ciclos de desenvolvimentos, teremos uma família composta de N *frameworks*.

Por fim, o último passo corresponde ao desenvolvimento de aplicações a partir do *framework*. Este passo envolve a escolha do *framework* que melhor atende aos requisitos da aplicação a ser gerada, dentre os pertencentes à família de *frameworks*. A escolha deve ser baseada nos requisitos levantados para a aplicação, na documentação e

no roteiro de utilização do *framework*, a fim de verificar em qual deles a aplicação se encaixa melhor. A maneira mais fácil de desenvolver as aplicações é utilizando os componentes já existentes no *framework*, assim, a aplicação reutiliza as interfaces e as regras para conectar os componentes.

4.3 Padrões utilizados na Construção do Framesec

Conforme citado na Seção 4.1, os padrões podem ser utilizados como base para a construção de *frameworks*. Sendo assim, para o desenvolvimento do Framesec proposto neste trabalho, utilizou-se como base a documentação da linguagem de padrões para softwares criptográficos *Tropyc* [77]. Esta linguagem descreve um conjunto de nove padrões de segurança: *Information Secrecy, Message Authentication, Message Integrity, Sender Authentication, Secrecy with Authentication, Secrecy with Signature, Secrecy with Integrity, Signature with Appendix*, e *Secrecy with Signature with Appendix*. Estes padrões são classificados de acordo com os quatros objetivos fundamentais da criptografia (confidencialidade, integridade, autenticação e não-repudiação, citados na Seção 2.1) e compõem um conjunto fechado de padrões para este domínio.

Na linguagem de padrões *Tropyc*, os padrões são definidos de acordo com os serviços criptográficos que as aplicações requisitam. Quando uma comunicação ou troca de informações através de arquivos é realizada, às vezes é necessário garantir o segredo ou confidencialidade das informações (*Information Secrecy*). Entretanto, somente a confidencialidade não previne a comunicação ou os arquivos de serem modificados ou substituídos. Particularmente na comunicação, garantir a integridade e autenticação da mensagem (*Message Integrity* e *Messagem Authentication*, respectivamente) é também importante. Existem contextos onde é necessário prevenir a entidade de negar suas ações ou confirmá-las. Por exemplo, alguma forma de autenticação do emissor (*Sender Authentication*) é necessária durante a realização de compras utilizando a Internet.

Em alguns casos, os requisitos de segurança das aplicações baseados em criptografia necessitam da composição de serviços básicos com a finalidade de prover confidencialidade com integridade (*Secrecy with Integrity*), confidencialidade com autenticação da mensagem (*Secrecy with Authentication*), confidencialidade com autenticação do emissor (*Secrecy with Signature*).

As operações criptográficas consomem recursos e gastam tempo de processamento, assim, a autenticação do emissor pode ser mais rápida se for realizada sobre um resumo dos dados a serem enviados (*Signature with Appendix*). Para maior eficiência, é recomendada que seja utilizado a confidencialidade e a autenticação do emissor sobre um resumo dos dados (*Secrecy with Signature with Appendix*), ao invés da confidencialidade com autenticação do emissor (*Secrecy with Signature*).

Todas as situações acima necessitam de uma arquitetura de software flexível e reutilizável. Esta arquitetura é comum para todos os padrões criptográficos definidos na linguagem de padrões *Tropyc* e são abstraídos de um meta-padrão criptográfico *Cryptographic Metapattern*.

A Figura 4.2 ilustra o grafo acíclico direcionado que representa os padrões da linguagem de padrões *Tropyc* e os relacionamentos existentes entre eles. Uma aresta partindo do padrão A para o padrão B informa que este último foi gerado a partir do padrão A. O *Cryptographic Metapattern* fornece uma micro-arquitetura orientada a objeto para os quatro padrões básicos da linguagem (*Message Authentication, Information Secrecy, Sender Authentication, Message Integrity*), sendo possível a sua utilização nos padrões gerados a partir destes padrões básicos.

O Framesec foi, portanto, definido através da documentação dos nove padrões de segurança *Tropyc*, além de utilizar a estrutura definida através do *Cryptographic Metapattern*. Durante a sua construção, que foi realizada utilizando a metodologia iterativa e incremental conforme citada na Seção 4.2, foi criada uma família de *framework* de segurança, contendo nove *frameworks*. A cada ciclo de desenvolvimento realizado, um padrão de segurança era adicionado, bem como as classes necessárias para a sua utilização.

Além dos padrões de segurança definidos na linguagem de padrões *Tropyc*, foram utilizados os padrões *Forwarder-Receiver* [30], o *Strategy* [31] e o *NullObject* [32].

O padrão *Forwarder-Receiver* introduz um modelo de comunicação ponto-a-ponto (*peer-to-peer*), que permite a separação dos remetentes (*Forwarders*) e receptores (*Receivers*), dos mecanismos de comunicação utilizados para a entrega e recebimento das informações, provendo assim uma comunicação interprocesso transparente.

A Figura 4.3 ilustra o padrão de projeto *Forwarder-Receiver*. Este padrão descreve 3 (três) componentes: *Forwarders*, *Receivers* e *Peers*. O componente *Peer* é

responsável pelas tarefas da aplicação. Para transmitir os resultados das tarefas executadas por este *Peer*, é necessária a comunicação com outro *Peer*. Este talvez esteja localizado em um processo diferente ou em uma máquina diferente. Cada *Peer* conhece o nome do *Peer* remoto com o qual necessita se comunicar. Este utiliza o *Forwarder* para enviar mensagens aos outros *Peers* e o *Receiver* para receber as mensagens dos outros *Peers*.

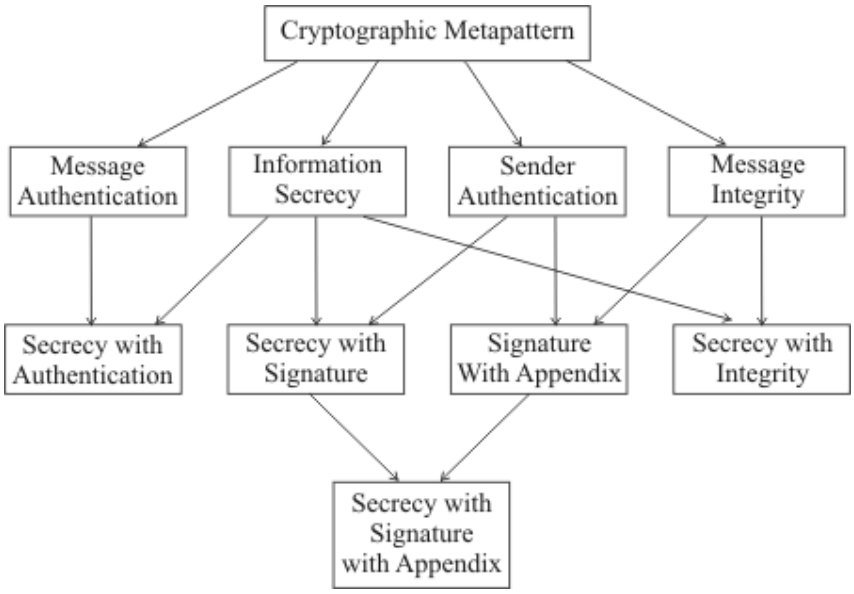

Figura 4.2. Padrões da linguagem de padrões *Tropyc* [77].

O *Forwarder* e *Receiver* utilizam um recipiente, chamado *Entry*, responsável por realizar o mapeamento dos nomes dos *Peers* para endereços reais. Antes de realizar a transmissão ou recebimento das mensagens, este recipiente é consultado com o intuito de recuperar a localização física do *Peer* remoto.

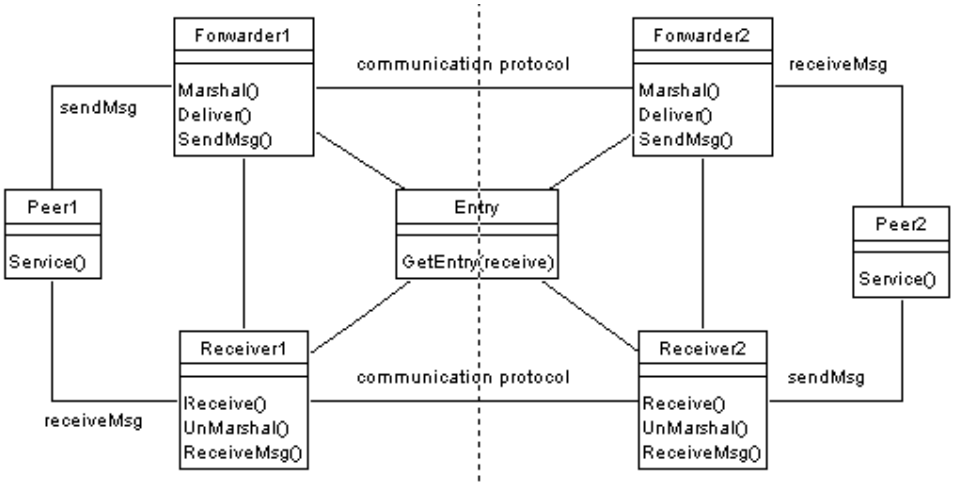

Figura 4.3. Padrão de projeto *Forwarder-Receiver* (adaptado de [30]).

Portanto, o padrão *Forwarder-Receiver* foi utilizado na construção do Framesec com a finalidade de realizar a separação dos pontos comunicantes das tarefas intrínsecas à comunicação, fornecendo assim uma comunicação interprocesso transparente.

O padrão *Strategy* é utilizado quando se quer ter manter a independência do algoritmo utilizado para realizar uma determinada tarefa. Sendo assim, este define uma família de algoritmos e os encapsula, permitindo ao objeto utilizador realizar a permuta entre os algoritmos sempre que este achar conveniente.

O *Strategy* foi utilizado na definição do Framesec com a finalidade de fornecer aos *Peers* (clientes) comunicantes, a possibilidade de mudança da estratégia de segurança utilizada para proteger a comunicação, sempre que os mesmos acharem necessário.

A Figura 4.4 ilustra o diagrama de classes do padrão *Strategy*. Cada estratégia de segurança (*ConcreteStrategy*) do Framesec corresponde a um padrão de segurança definido na *Tropyc*. Isto é, a cada ciclo de desenvolvimento, uma estratégia correspondente ao padrão de segurança (e.g., *IntegrityStrategy, SecrecyStrategy*) era adicionada ao Framesec.

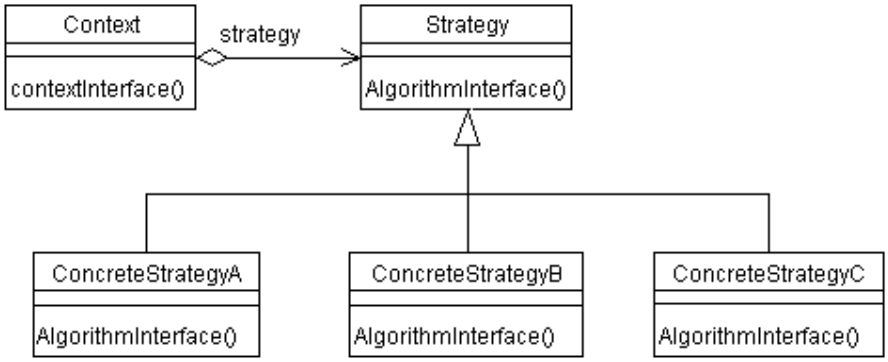

Figura 4.4. Diagrama de classes do padrão Strategy (adaptado de [31]).

Por último, o padrão *NullObject* é utilizado quando se quer eliminar a verificação de nulo (*null checking*) do objeto no lado cliente. Este padrão implementa todas as operações do objeto real, mas estas não realizam qualquer tarefa (*nothing*). A Figura 4.5 ilustra o diagrama de classes do padrão.

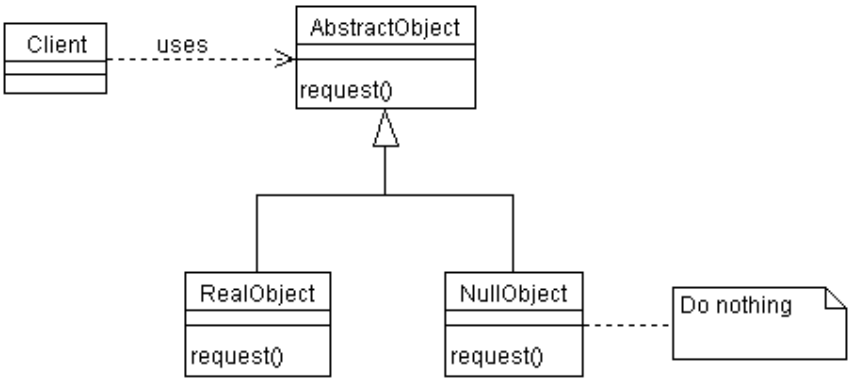

Figura 4.5. Diagrama de classes do padrão NullObject (adaptado de [32]).

O padrão *NullObject* foi utilizando na definição do Framesec para eliminar a verificação de uma estratégia de segurança nula no cliente (*Peer*), ou seja, uma estratégia que fornece às aplicações uma transformação de segurança nula foi adicionada ao Framesec, permitindo a transmissão de informações sem qualquer criptografia sobre os dados (i.e., em texto plano).

Na próxima seção será apresentado o *framework* para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel proposto neste trabalho.

4.4 O Framesec

O Framesec é um *framework* de aplicação com domínio fechado aos serviços de segurança, que oferece às aplicações no ambiente de computação móvel suporte aos quatro objetivos da criptografia citados na Seção 2.1, bem como às suas combinações, através da reutilização da sua estrutura e implementação para a construção de mecanismos para a provisão de segurança fim-a-fim.

Durante a construção do Framesec, foram executados nove ciclos de desenvolvimento seguindo os passos definidos na Seção 4.2. O primeiro passo da metodologia de construção de *frameworks* iterativa e incremental corresponde ao passo "Análise de Requisitos". Sendo assim, o segundo passo (Passo 2) corresponde ao primeiro ciclo de desenvolvimento do *framework*, o terceiro passo (Passo 3) ao segundo ciclo e assim sucessivamente. Ao finalizar o nono ciclo de desenvolvimento do Framesec, terá sido executado o décimo passo (Passo 10) da metodologia. O último passo (Passo 11) a ser executado corresponde ao passo "Instanciar Aplicações", que finaliza o processo de construção do Framesec com onze passos executados.

A cada ciclo, um novo *framework* foi gerado contendo as funcionalidades de segurança do *framework* definido no ciclo anterior, além dos serviços de segurança do padrão da *Tropyc* adicionado no ciclo corrente. Serão apresentados os seguintes passos executados para a construção do Framesec: o Passo 1, correspondente à análise de requisitos do *framework*; o Passo 2, correspondente ao primeiro ciclo de desenvolvimento; o Passo 6, que corresponde a um ciclo intermediário de desenvolvimento do Framesec; o Passo 10 que executa o último ciclo de desenvolvimento do Framesec e o Passo 11, que demonstra a utilização do *framework* ao "Instanciar Aplicações". Desta forma, fica demonstrada a evolução do Framesec nos vários ciclos de desenvolvimento, até atingir o seu estágio final. Os outros passos, correspondentes aos ciclos restantes, são desenvolvidos de forma semelhante aos passos 2 e 6 apresentados aqui.

A Figura 4.6 ilustra a evolução do Framesec nos vários ciclos de seu desenvolvimento, bem como as funcionalidades de segurança adicionadas a cada iteração do processo de construção.

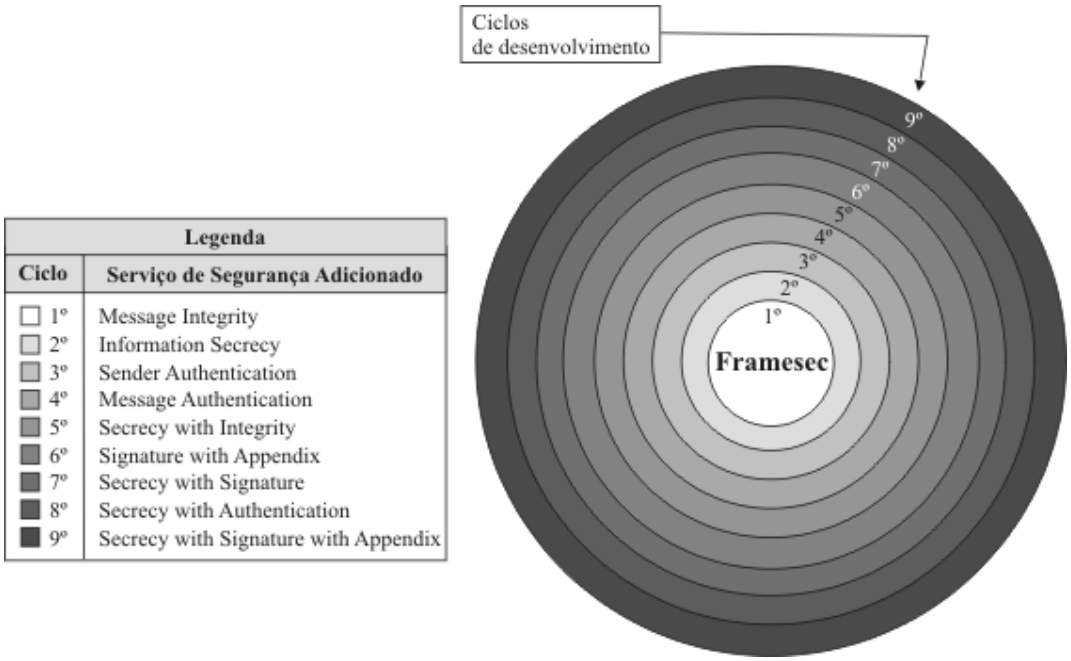

Figura 4.6. Os ciclos de desenvolvimento do Framesec.

No primeiro ciclo de desenvolvimento, o Framesec foi construído de forma a disponibilizar o serviço de integridade da mensagem, conforme a documentação do padrão *Message Integrity* da *Tropyc*. No segundo ciclo, foram adicionadas as funcionalidades de segurança para a provisão da confidencialidade da informação, conforme a documentação do padrão *Information Secrecy* da *Tropyc*. Portanto, o 2º

Framesec construído disponibiliza tantos os serviços de segurança adicionados por este ciclo, como os definidos nos requisitos do ciclo anterior.

As funcionalidades de segurança dos *frameworks* construídos nos ciclos seguintes correspondem ao serviço de segurança adicionado ao ciclo corrente, somados aos serviços definidos nos ciclos anteriores. Sendo assim, no último ciclo de desenvolvimento, teremos a versão final do Framesec contendo todos os serviços de segurança documentos pela linguagem de padrões *Tropyc*, compondo um *framework* de aplicação de domínio fechado específico de segurança.

Para a representação dos diagramas e demais modelos necessários à definição do Framesec, foi utiliza a notação UML - *Unified Model Language* [38]. Esta notação é largamente utilizada para a modelagem de aplicações orientadas a objeto, bem como para a modelagem de *frameworks* (pois a sua estrutura é orientada a objeto).

As instâncias do Framesec são implementadas na plataforma J2ME MIDP/CLDC 1.0 [71], assim como os algoritmos de criptografia utilizados para compor os *hot spots* do *framework*. O motivo para a utilização desta plataforma de desenvolvimento ao invés de outras, tais como C, BREW e Personal Java, é que esta garante a portabilidade das aplicações com uma vasta classe de DMs, tais como Palms, telefones móveis e Pocket PCs. Por exemplo, uma aplicação desenvolvida nesta plataforma para ser hospedada em DMs Palms, pode ser facilmente portada para telefones móveis que apresentam suporte a esta plataforma.

Os algoritmos criptográficos, por sua vez, são implementados e distribuídos por um grupo de pesquisa que desenvolve APIs Java na área de segurança, chamado *Legion of the Bouncy Castle* [96]. Estas foram utilizadas, na versão 1.24, por serem as únicas implementadas na plataforma J2ME disponíveis e distribuídas livremente à comunidade científica. A próxima seção apresenta os ciclos (1º, 5º e último ciclo) de desenvolvimento do Framesec.

4.4.1 Desenvolvimento do Framesec

Os passos seguintes compõem o processo proposto na Seção 4.2 e conduzem o desenvolvimento do Framesec. Conforme citado na seção anterior, serão demonstrados os Passos 1, 2, 6, 10 e 11, executados durante o processo de desenvolvimento do Framesec.

Passo 1: Análise do Domínio

a) Definição do escopo

Objetivo:

- Prover uma estrutura reutilizável para a criação de aplicações que possuem requisitos de segurança, no ambiente de computação móvel.

Limites:

- O *framework* não fornecerá suporte a mecanismos de troca chaves, bem como para o armazenamento seguro. Também não será considerado o mecanismo que deve ser executado previamente para definir os algoritmos de criptografia que serão utilizados pelas partes comunicantes (i.e., DM e o SA).

Benefícios:

- Economia de tempo de construção de aplicações que possuem requisitos de segurança fim-a-fim no ambiente de computação móvel;
- Permite a extensão de classes obedecendo a novos requisitos de segurança, a partir dos serviços disponibilizados pelo *framework*.

b) Modelo Estático preliminar

A arquitetura geral do *Framesec* foi definida a partir do *Cryptographic Metapattern* definido na linguagem de padrões *Tropyc,* combinada ao padrão *Forwarder-Receiver*, que provê a comunicação interprocesso transparente, ambos citados na Seção 4.3.

Em uma comunicação de dados estão envolvidos dois pontos, ou *Peers*. Cada *Peer* possui um *Forwarder* e um *Receiver*, responsáveis pelas operações de envio e recebimento, respectivamente. Um *Peer* pode se comunicar com um ou vários *Peers* simultaneamente. Para isso, é necessário manter um repositório contendo as localizações físicas dos *Peers* comunicantes, de forma a permite a comunicação interprocesso transparente.

O *Forwarder* e *Receiver* utilizam o repositório *Entry*, que é responsável pelo mapeamento de nomes de *Peers* para endereço reais. Além disso, este repositório permite adicionar, mover ou deletar endereços dos *Peers* dinamicamente. A cada entrada neste repositório, são informados o *Codifier* e o *Decodifier* a serem utilizados posteriormente pelo *Forwader* e *Receiver*.

Durante o processo de envio, o *Forwarder* consulta o repositório *Entry*, recuperando a localização física do *Peer* comunicante, além do *Codifier* que será utilizado para realizar as transformações criptográficas sobre os dados a serem enviados, conforme os requisitos de segurança definidos pela aplicação. Por sua vez, o *Receiver* também consulta este repositório e recupera as informações do *Peer* do qual receberá os dados transmitidos. Este, ao receber os dados da transmissão, solicita ao *Decodifier* a realização do processo inverso de codificação, de forma a recuperar a mensagem enviada. O *Codifier*, assim como o *Decodifier*, utilizam os algoritmos criptográficos (*Cryptographic Algorithm*) para realizar a encriptação e decriptação das informações.

É necessária uma implementação específica do *Forwarder*, *Receiver* e *Entry*, para cada protocolo de comunicação utilizado (e.g., http) para a transmissão entre os DMs e o SA. A Figura 4.7 abaixo ilustra o modelo de classes preliminar do *Framesec*.

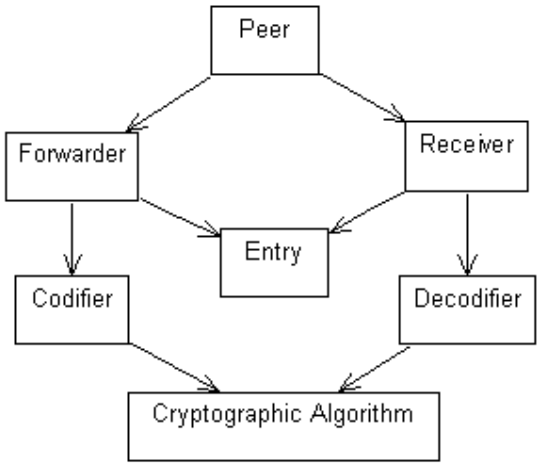

Figura 4.7. Modelo estático preliminar do Framesec.

Passo 2: 1º Ciclo de Desenvolvimento

Ao final deste passo, teremos o 1º *framework* (primeira versão do Framesec) construído, dentre a família de *frameworks* que será gerada.

Passo 2.1: Coleta de Requisitos

O *framework* consiste de uma estrutura reutilizável, que provê integridade das mensagens (*Message Integrity*) trocadas entre o DM e o SA.

a) Requisitos Funcionais

- Prover integridade da mensagem;

– Fornecer comunicação interprocesso transparente;
– Fornecer funcionalidades de comunicação que utilizam o protocolo HTTP;
– Permitir a comunicação com vários *Peers* simultaneamente.

b) Requisitos Não-Funcionais

– Ser extensível, possibilitando adicionar novos *Modification Detection Code* (MDCs ou função hash, conforme o Apendice B.3) para prover a integridade;
– Ser de fácil compreensão;
– Conter documentação.

c) Descrição dos Casos de Uso

01 – Caso de Uso Abstrato Enviar Mensagem

Descrição: é calculado o MDC da mensagem a ser enviada utilizando o algoritmo criptográfico (função *hash*, ou *digest*) combinado entre as partes comunicantes. A mensagem, assim como o MDC calculado, serão enviados ao *Peer* de destino.

02 – Caso de Uso Abstrato Codificar Mensagem

Descrição: é calculado o MDC da mensagem a utilizando o algoritmo criptográfico (função *hash*, ou *digest*) desejado. A mensagem, assim como o MDC calculado, são retornados ao objeto que requisitou a codificação.

03 – Caso de Uso Abstrato Receber Mensagem

Descrição: é realizada a separação do MDC enviado da mensagem recebida. A partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC enviado e o calculado, a fim de verificar se não houve modificações durante a transmissão. Se forem iguais, o recebimento foi realizado com sucesso e a integridade da mensagem estará garantida.

04 – Caso de Uso Abstrato Decodificar Mensagem

Descrição: é realizada a separação do MDC da mensagem. A partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC passado e o calculado, a fim de verificar se não houve modificações. Se forem iguais, a integridade estará verificada.

Passo 2.2: Análise e Detecção de Hot Spots

a) Análise

No diagrama de classes da análise são adicionadas as classes *ForwarderHTTP*, *ReceiverHTTP*, *EntryHTTP* (responsáveis pelas funcionalidades de comunicação utilizando o protocolo HTTP), *CodifierStrategy*, *DecodifierStrategy, IntegrityStrategy* (responsáveis pelas tarefas de codificação e decodificação das informações, oferecendo a integridade da mensagem), *Parameter* e *IntegrityParams* (que fornece à classe *IntegrityStrategy* as informações necessárias para a execução das operações criptográficas). A Figura 2.1 ilustra o diagrama de classes do primeiro ciclo de desenvolvimento.

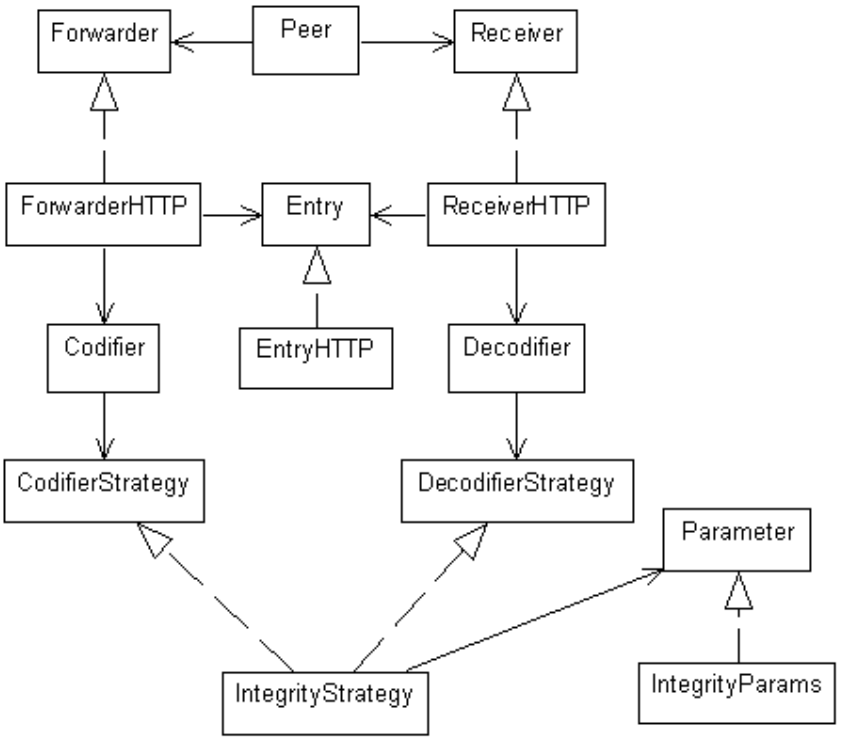

Figura 4.8. Diagrama de Classes do 1º Ciclo de desenvolvimento

b) Detecção de *Hot Spots*

Número: 1

Nome: Variabilidade do algoritmo MDC

Descrição1: Deve permitir a utilização de qualquer um dos algoritmos MDC (i.e., funções *hash,* conforme o Apêndice B.3) existente no pacote de APIs criptográficas *BouncyCastle*.

Exemplos de utilização: MD2, MD4, MD5, RIPEMD128, RIPEMD160, SHA 256, SHA 384, SHA 512, SHA-1 e Tiger.

Tipo de adaptação requerida: 1 – Habilitar uma característica (conforme descrito na Seção 4.2).

Grau de apoio: Suporte padrão

Descrição2: Deve permitir adicionar novos algoritmos do tipo MDC, ou implementações diferentes do mesmo algoritmo.

Exemplo de utilização: Utilizar um novo algoritmo criado pela comunidade científica.

Tipo de adaptação requerida: 5 – Adicionar uma característica

Grau de apoio: Ilimitado

Passo 2.3: Projeto

É uma boa prática de projeto separar no *framework* as interfaces das implementações abstratas. Isto resulta em um *framework* com camadas claramente estruturadas, onde as camadas mais altas (mais abstratas) são independentes das camadas mais baixas (mais concretas).

Assim as camadas mais baixas podem ser substituídas sem requerer mudanças nas camadas mais altas. A Figura 4.9 ilustra a estrutura do *framework* dividido em três camadas: a camada de interface, a camada de implementação de núcleo e camada padrão.

A camada de interface (classes na cor branca) prover a reutilização do projeto. É a mais estável das camadas, portanto, não possui implementação. Por sua vez, a camada núcleo (classes na cor cinza claro) provê a reutilização, tanto ao nível de projeto quanto de código. Devido a essas características, ela é menos estável que a camada de interface. Por fim, a camada padrão (classes na cor cinza escuro) provê a reutilização de implementação, sendo a menos estável de todas as camadas.

a) Definição de Padrões de Projeto

Problema: prover a integridade da mensagem, através da utilização de algoritmos MDC.

Padrão: *Message Integrity* [77]

Problema: para prover a integridade da mensagem, podem ser utilizadas implementações de algoritmos MDC diferentes.

Padrão: *Strategy* [31]

Problema: durante a comunicação pode ser necessária à utilização de uma transformação nula, ou seja, nenhum mecanismo de segurança será aplicado sobre as informações transmitidas.

Padrão: *NullObject* [32]

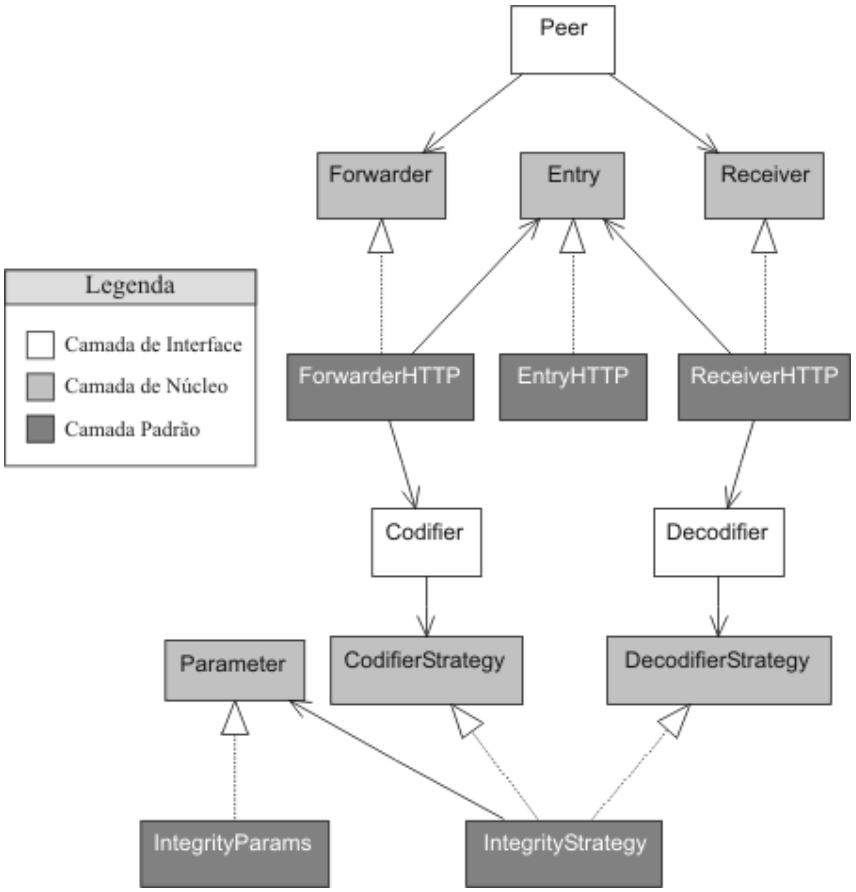

Figura 4.9. Estrutura do *framework* no modelo de três camadas.

Problema: para cada tipo de protocolo de comunicação utilizado, uma implementação diferente de *Forwarder*, *Receiver* e *Entry* são necessárias.

Padrão: *Strategy* [31]

Problema: para cada tipo estratégia, uma implementação diferente de *Parameter* é necessária.

Padrão: *Strategy* [31]

b) Diagrama de Classes do Projeto

A Figura 4.10 ilustra o diagrama de classes de projeto do *framework* definido neste ciclo. Este diagrama corresponde à definição da estrutura do primeiro Framesec, que apresenta suporte ao serviço de integridade da mensagem, conforme citado na Seção 4.4. A este diagrama, foram adicionadas as classes *IntegrityStrategy,* é responsável pela ligação do Framesec com os algoritmos criptográficos pertencentes ao pacote de API *Bouncy Castle*, a classe *NullStrategy*, que realiza a transformação nula de codificação e decodificação da mensagem e a classe *Parameter*, que mantém os parâmetros necessários à estratégia utilizada. A Figura 4.11 ilustra o diagrama de classes detalhado do primeiro Framesec.

Passo 2.4: Implementação e instanciação Exemplo

Conforme citado anteriormente, a plataforma de desenvolvimento utilizada para implementar o Framesec é a J2ME MIDP/CLDC 1.0. A seguir, serão apresentadas instâncias exemplo deste Framesec implementado.

a) Instanciação Exemplo

Antes de realizar a instanciação do Framesec, o desenvolvedor deve decidir se deseja ou não instanciar o *framework* por completo, isto é, instanciar toda a estrutura de comunicação interprocesso transparente bem como a estrutura de codificação (*Codifier*) e decodificação (*Decodifier*) das mensagens. Caso contrário, a instância do Framesec pode ser gerada a partir do *Codifier* e do *Decodifier*, ficando a cargo do desenvolvedor de aplicações estabelecer a conexão com a outra parte comunicante (i.e., SA).

Além da decisão quanto à estrutura do *framework* a ser instanciada, o desenvolvedor deverá decidir qual a função *hash* (veja a Seção B.3) que será utilizado para garantir a integridade das mensagens. Este pode utilizar a ferramenta PEARL, proposta neste trabalho e descrita no Capítulo 0, para realizar uma avaliação de desempenho das funções *hash* no dispositivo móvel para qual a aplicação se destina. Nos exemplos abaixo, é suposto que o algoritmo escolhido conforme os parâmetros de desempenho das funções *hash* avaliadas pela ferramenta PEARL é o MD5.

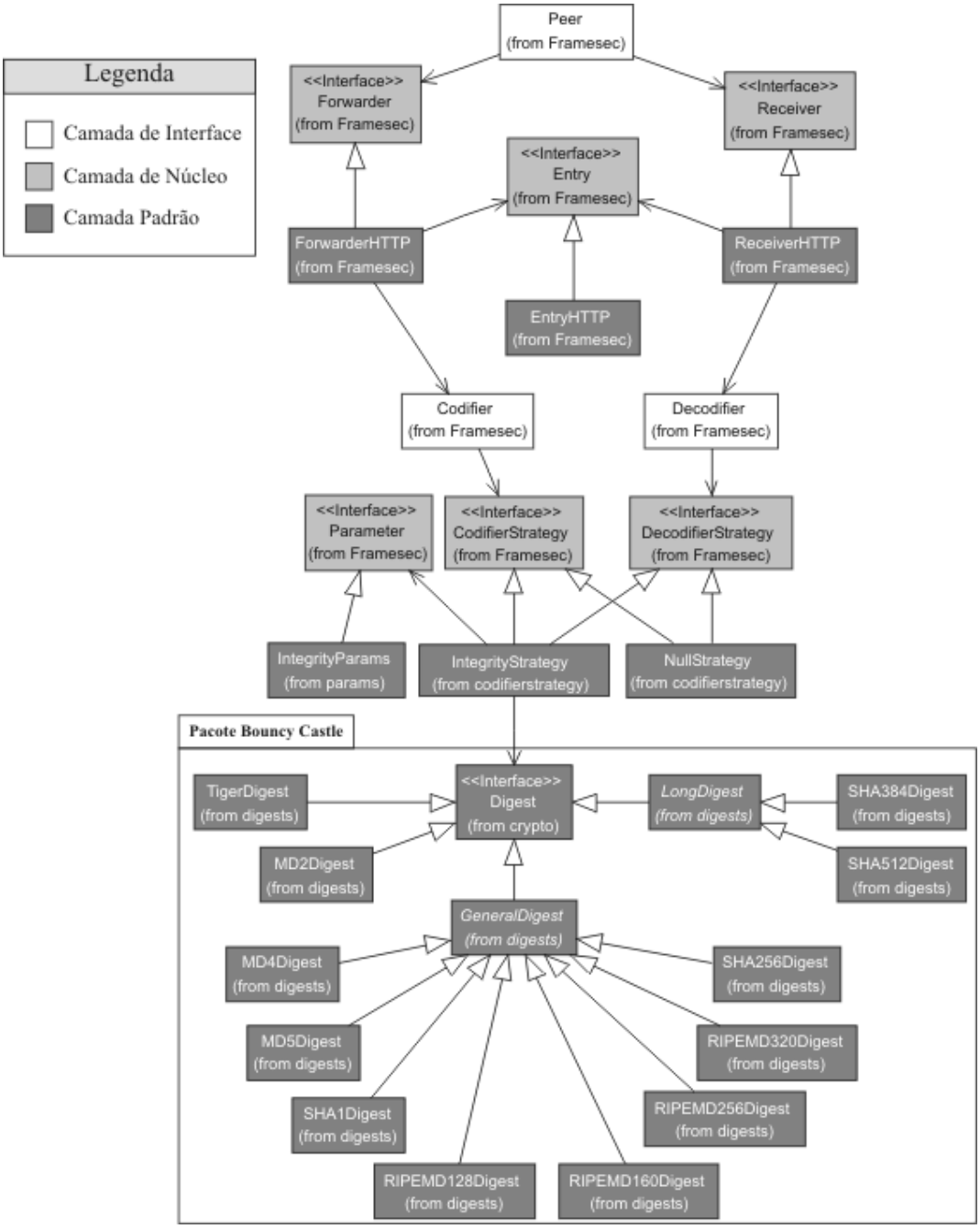

Figura 4.10. Diagrama de classe do primeiro Framesec.

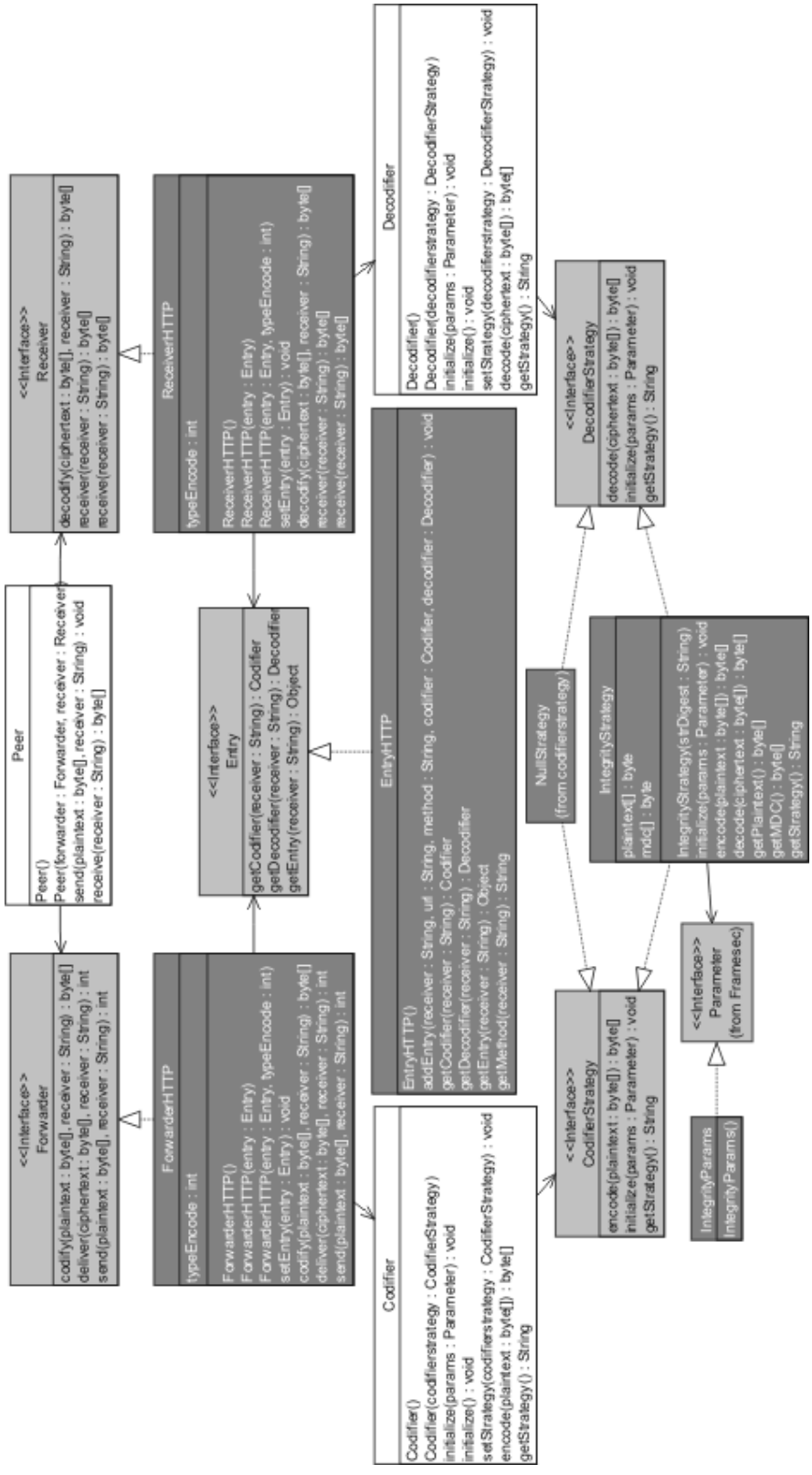

Figura 4.11. Diagrama de classes detalhado do primeiro Framesec.

A Figura 4.12 e a Figura 4.13 ilustram a instanciação completa do Framesec e a instanciação a partir do *Codifier* e *Decodifier*, respectivamente:

```
/* Esta forma de utilização instancia todo o framework, incluindo a estrutura de
comunicação interprocesso transparente. */
// Criando o codificador e decodificador da mensagem

Peer alice;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext;

codifier = new Codifier(new IntegrityStrategy("MD5"));
decodifier = new Decodifier(new IntegrityStrategy("MD5"));

// A estratégia de integridade da mensagem não necessita de parâmetros de
// inicialização

codifier.initialize();
decodifier.initialize();
peers = new EntryHTTP();

// Alice se comunica com Bob, então devem ser informados os dados referentes à Bob.
// Este método é executado a cada novo ponto de comunicação estabelecido por Alice

peers.addEntry("Bob", "http://200.17.37.12:8080", "post", codifier, decodifier);
alice = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.12. Exemplo de instanciação completa do Framesec.

```
/* Esta forma de utilização não instancia a estrutura de comunicação interprocesso
transparente. */
// Criando o codificador e decodificador da mensagem

Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext;
codifier = new Codifier(new IntegrityStrategy("MD5"));
decodifier = new Decodifier(new IntegrityStrategy("MD5"));
```

```
// A estratégia de integridade da mensagem não necessita de parâmetros de
// inicialização

codifier.initialize();
decodifier.initialize();
...

/* o plaintext possui o valor da mensagem a ser codificada. O ciphertext, por sua vez,
recebe o resultado da codificação */
ciphertext = codifier.encode(plaintext);  // ciphertext = msg + MDC(plaintext)
...
// Enviar a mensagem
...
/* após receber a mensagem, o processo de codificação inversa deve ser realizado */
plaintext = decodifier.decode(ciphertext);
```

Figura 4.13. Instanciação do Framesec, a partir do *Codifier* e *Decodifier*.

Passo 2.5: Elaboração do Roteiro de Utilização

Objetivo: Descrever os passos que devem ser seguidos pelo desenvolvedor para construir aplicações para DMs que possuem requisitos de integridade de mensagem, a partir das instâncias do Framesec.

a) Instanciando o Framesec

Para instanciar o Framesec e utilizar a estratégia *IntegrityStrategy* para garantir a integridade das informações, o desenvolvedor deve decidir qual o algoritmo MDC será utilizado, conforme mencionado anteriormente, de acordo com a Tabela 4.2.

Tabela 4.2. Algoritmos MDC disponíveis para a *IntegrityStrategy*.

Algoritmos MDC (função Hash) disponíveis na IntegrityStrategy	
Nome do algoritmo (Framesec)	**Classe Concreta (implementação Bouncy Castle)**
MD2	MD2Digest
MD4	MD4Digest
MD5	MD5Digest
RIPEMD128	RIPEMD128Digest
RIPEMD160	RIPEMD160Digest
SHA-256	SHA256Digest
SHA-384	SHA384Digest
SHA-512	SHA512Digest
SHA-1	SHA1Digest
Tiger	TigerDigest

Além desta decisão, o desenvolvedor deve optar por instanciar o Framesec por completo (que fornece a comunicação interprocesso transparente) ou não. Caso o

desenvolvedor não deseje a instanciação por completo, as classes *Peer*, *Forwarder*, *Receiver* e *Entry* não serão instanciadas, cabendo ao desenvolvedor realizar o estabelecimento da conexão com o ponto remoto (i.e., o SA) da comunicação. A Figura 4.12 e a Figura 4.13 ilustram as duas formas de instanciação citadas acima.

b) Utilizando um novo algoritmo MDC

Caso o desenvolvedor deseje utilizar um novo algoritmo MDC, este deve modificar a classe *IntegrityStrategy* a fim de torná-lo visível às instâncias do Framesec. Especificamente, deve modificar o método construtor desta classe, conforme a Figura 4.3.

c) Utilizando um novo protocolo de comunicação

Caso o desenvolvedor deseje utilizar um novo protocolo de comunicação, este deve implementar as classes *ForwarderNewProtocol*, *ReceiverNewProtocol* e *EntryNewProtocol*, a partir das interfaces *Forwader*, *Receiver* e *Entry*, respectivamente.

```
// Construtor da estratégia que realiza a integridade da mensagem
public IntegrityStrategy (String strDigest) {

  // Identificando o algoritmo a ser utilizado

  if (strDigest.equals("MD2"))            {digest = new MD2Digest();}
  if (strDigest.equals("MD4"))            {digest = new MD4Digest();}
  if (strDigest.equals("MD5"))            {digest = new MD5Digest();}
  if (strDigest.equals("RIPEMD128")) {digest = new RIPEMD128Digest();}
  if (strDigest.equals("RIPEMD160")) {digest = new RIPEMD160Digest();}
  if (strDigest.equals("SHA-256"))   {digest = new SHA256Digest();}
  if (strDigest.equals("SHA-384"))   {digest = new SHA384Digest();}
  if (strDigest.equals("SHA-512"))   {digest = new SHA512Digest();}
  if (strDigest.equals("SHA-1"))          {digest = new SHA1Digest();}
  if (strDigest.equals("Tiger"))          {digest = new TigerDigest();}
  /* O desenvolvedor deve adicionar, neste ponto, uma nova entrada para o algoritmo
adicionado */
}
```

Figura 4.14. Método construtor da classe *IntegrityStrategry*.

Com o término deste passo, o 1º ciclo de desenvolvimento do Framesec foi finalizado. O *framework* definido neste passo fornece, portanto, o serviço de integridade da mensagem às aplicações construídas a partir das instâncias do Framesec. Será

apresentado agora o sexto passo executado para a construção do Framesec, conforme citado na Seção 4.4.

Passo 6: 5º Ciclo de Desenvolvimento

Ao final deste passo, teremos o 5º *framework* construído, dentre a família de *frameworks* gerada, que fornece as funcionalidades de *Information Secrecy, Message Authentication, Message Integrity, Sender Authentication* e *Secrecy with Integrity*, conforme citado na Seção 4.4.1. As funcionalidade de segurança adicionadas neste ciclo é fornecida através da classe *SecrecyIntegrityStrategy* adicionada ao diagrama de classes, como será visto mais adiante.

Passo 6.1: Coleta de Requisitos

a) Requisitos Funcionais

- Prover confidencialidade com integridade da mensagem (*Secrecy with Integrity*).

b) Requisitos Não-Funcionais

- Ser extensível, possibilitando adicionar novos algoritmos de criptografia simétrica, assimétrica e funções *hash* (veja o Apêndice B) para prover a confidencialidade com integridade;
- Ser de fácil compreensão;
- Conter documentação.

c) Descrição dos Casos de Uso

01 – Caso de Uso Abstrato Enviar Mensagem

Descrição: é calculado o MDC da mensagem a ser enviada utilizando a função *hash* combinada entre as partes comunicantes. A mensagem concatenada com o MDC calculado é encriptada utilizando o algoritmo de criptografia e chaves combinadas entre as partes comunicantes, sendo posteriormente enviada ao *Peer* de destino.

02 – Caso de Uso Abstrato Codificar Mensagem

Descrição: é calculado o MDC da mensagem utilizando a função *hash* desejada. A mensagem concatenada com o MDC calculado é encriptada utilizando o algoritmo de criptografia e as chaves desejadas, sendo posteriormente retornada ao objeto que requisitou a codificação.

03 – Caso de Uso Abstrato Receber Mensagem

Descrição: é realizada a decriptação da mensagem recebida utilizando o algoritmo de criptografia e as chaves combinadas entre as partes comunicantes. Após isso, a mensagem é separada do MDC enviado. A partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC enviado e o calculado, a fim de verificar se não houve modificações durante a transmissão. Se forem iguais, o recebimento foi realizado com sucesso e a integridade da mensagem estará garantida.

04 – Caso de Uso Abstrato Decodificar Mensagem

Descrição: é realizada a decriptação da mensagem utilizando o algoritmo de criptografia e as chaves desejadas. Após isso, a mensagem é separada do MDC. A partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC passado e o calculado, a fim de verificar se não houve modificações. Se forem iguais, a integridade da mensagem estará garantida.

Passo 6.2: Análise e Detecção de Hot Spots

a) Análise

A classe *SecrecyIntergrityStrategy* é adicionada ao diagrama de classes da análise, em relação ao diagrama de classes definido no quarto ciclo (Passo 5) de desenvolvimento do Framesec. Esta classe é responsável pelo fornecimento do serviço de segurança confidencialidade com integridade (padrão *Secrecy with Integrity*). A Figura 4.15 ilustra o diagrama de classes definido neste passo.

b) Detecção de *Hot Spots*

Neste ciclo de desenvolvimento não foram detectados *hot spots*. Isto por que a estratégia adicionada para o provimento do serviço de confidencialidade com integridade (*SecrecyIntegrityStrategy*) utiliza duas estratégias previamente definidas, a *SecrecyStrategy* e *IntegrityStrategy*. Estas duas estratégias fornecem os *hot spots* que permitem adicionar novos algoritmos de criptografia, sendo assim, a estratégia *SecrecyIntegrityStrategy* herdará esta flexibilidade.

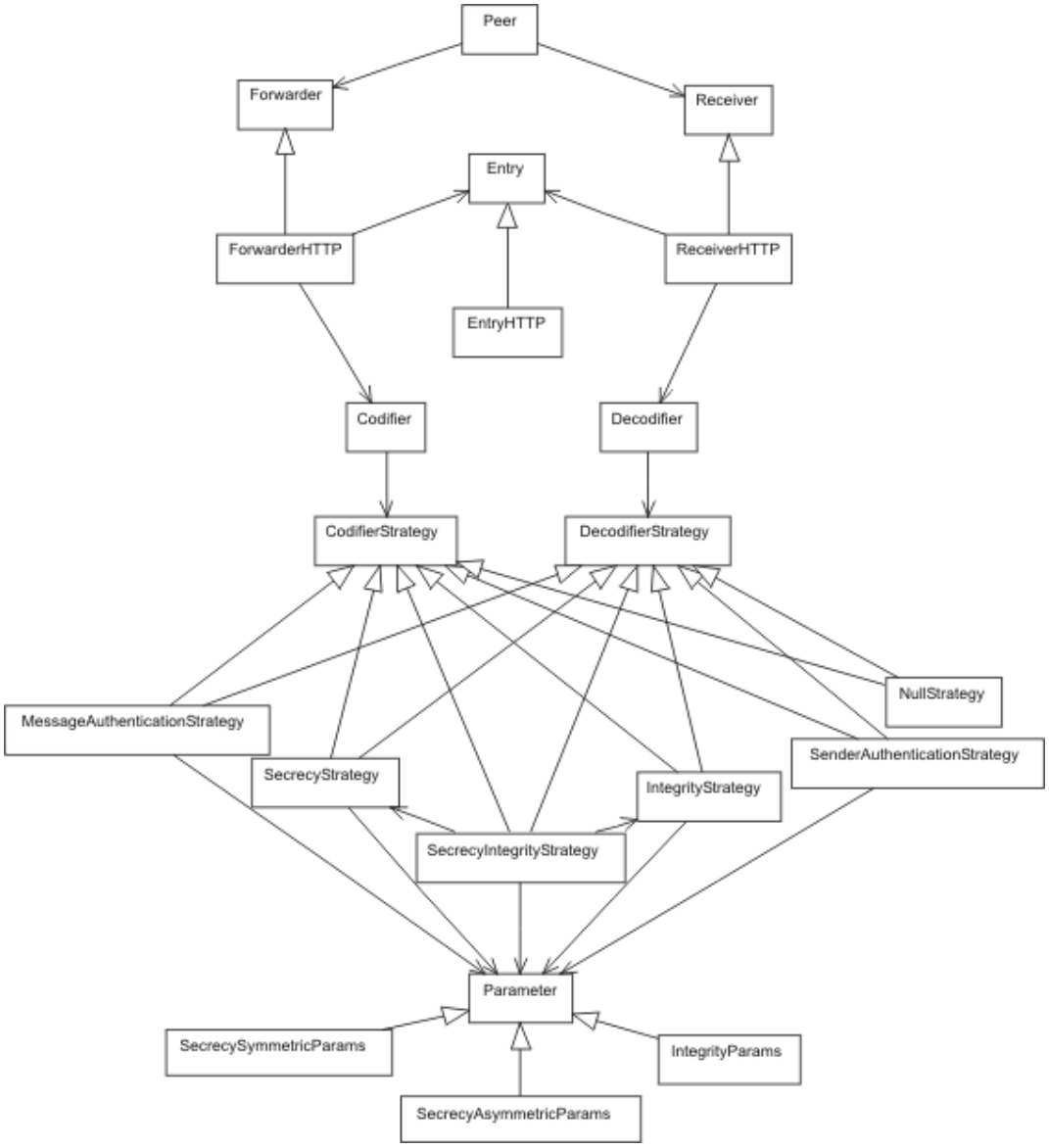

Figura 4.15. Diagrama de Classes da análise no 5º Ciclo de desenvolvimento

Passo 6.3: Projeto

a) Definição de Padrões de Projeto

Para a adição da estratégia *SecrecyIntegrityStrategy* não foram necessárias a utilização de novos padrões (utilizou o padrão *Strategy* já adicionado no Passo 2 do processo de desenvolvimento do Framesec).

b) Diagrama de Classes do Projeto

Este diagrama corresponde à definição da estrutura do quinto Framesec, que apresenta suporte ao serviço de integridade da mensagem (*IntegrityStrategy*),

autenticação do emissor (*SenderAuthenticationStrategy*), confidencialidade da informação (*SecrecyStrategy*), autenticação da mensagem (*MessageAuthenticationStrategy*) e confidencialidde com integridade (*SecrecyIntegrityStrategy*). A Figura 4.16 ilustra o diagrama de classes completo deste ciclo. Nesta figura, as classes do pacote *Bouncy Castle* foram suprimidas com o intuito de manter a legibilidade do diagrama. No entanto, a Figura 4.17, a Figura 4.18, a Figura 4.19 e a Figura 4.20, apresentam separadamente os diagramas de classes das estratégias *IntegrityStrategy*, *SenderAuthenticationStrategy, SecrecyStrategy* e *MessageAuthenticationStrategy*, respectivamente, juntamente com as classes do pacote *Bouncy Castle* utilizadas por cada estratégia.

Passo 6.4: Implementação e instanciação Exemplo

a) Instanciação Exemplo

O processo de instanciação do *framework* é o mesmo que foi detalhado no passo 2.4 do primeiro ciclo de desenvolvimento do Framesec. No entanto, o desenvolvedor deve decidir por qual estratégia de segurança utilizar (i.e., *IntegrityStrategy*, *SecrecyStrategy*) e quais os algoritmos criptográficos irão compor os *hot spots* do Framesec.

A primeira escolha a ser feita pelo desenvolvedor é a estratégia de segurança que será utilizada. De acordo com a estratégia escolhida, o desenvolvedor deverá utilizar os sub-módulos de avaliação da ferramenta PEARL, conforme descrito da Seção 5.3.2, para identificar os algoritmos que possuem o melhor desempenho no DM analisado. Para a estratégia *IntegrityStrategy*, os algoritmos disponíveis ao Framesec implementado em J2ME são listados na Tabela 4.2, já apresentada no passo 2.4. Por sua vez, as estratégias *SecrecyStrategy*, *SenderAuthentication* e *MessageAuthentication* permitem ao desenvolvedor utilizar os algoritmos listados na Tabela 4.3, na Tabela 4.4 e na Tabela 4.5, respectivamente.

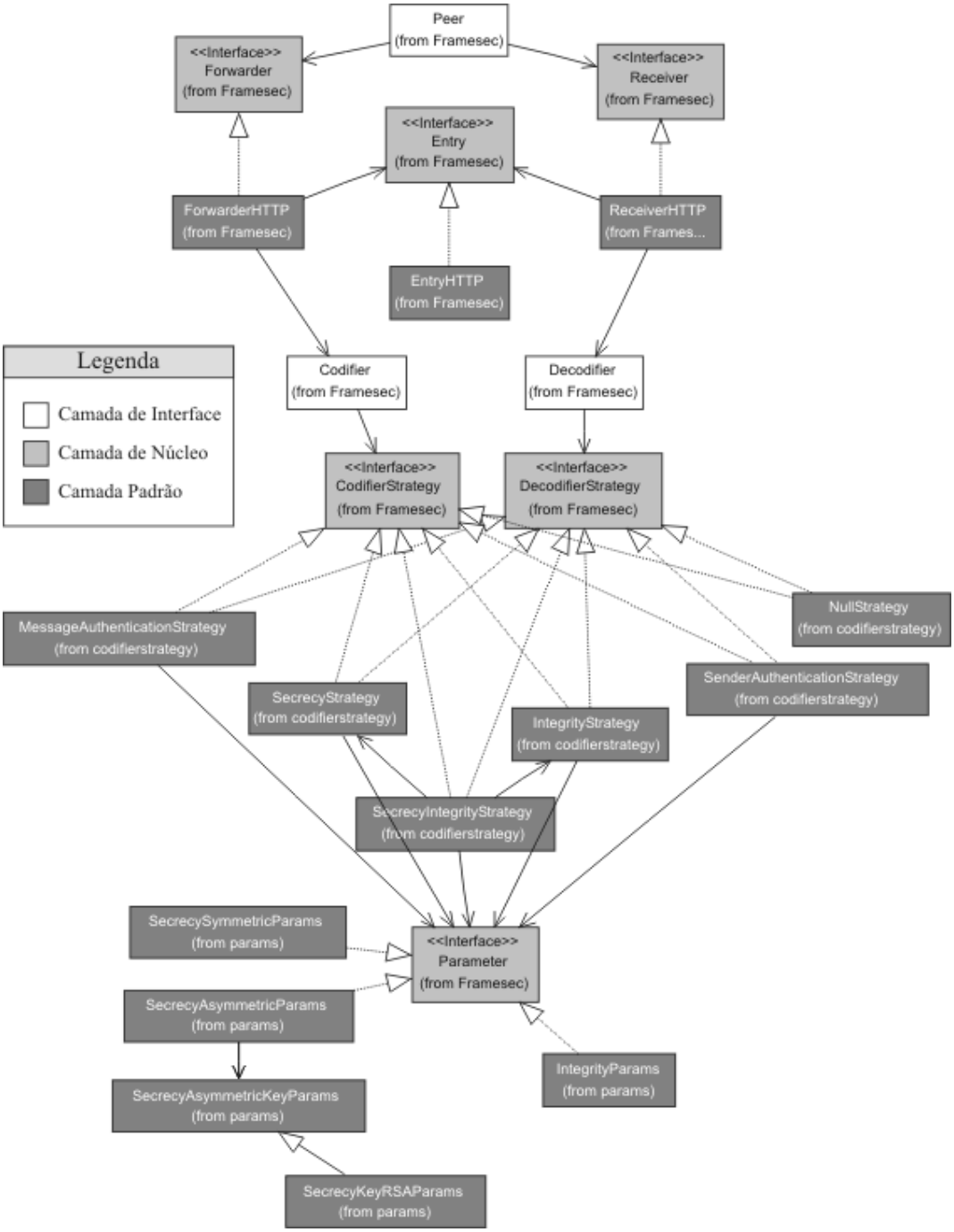

Figura 4.16. Diagrama de classe do quinto Framesec.

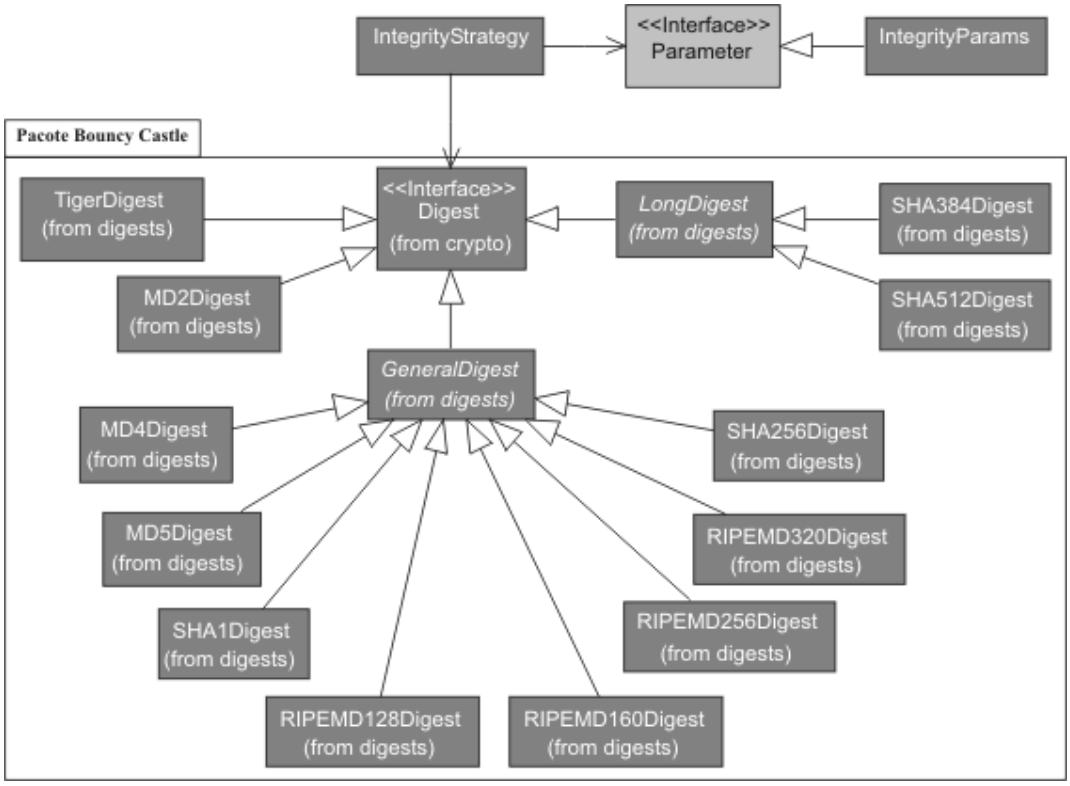

Figura 4.17. Diagrama de classes da estratégia *IntegrityStrategy*.

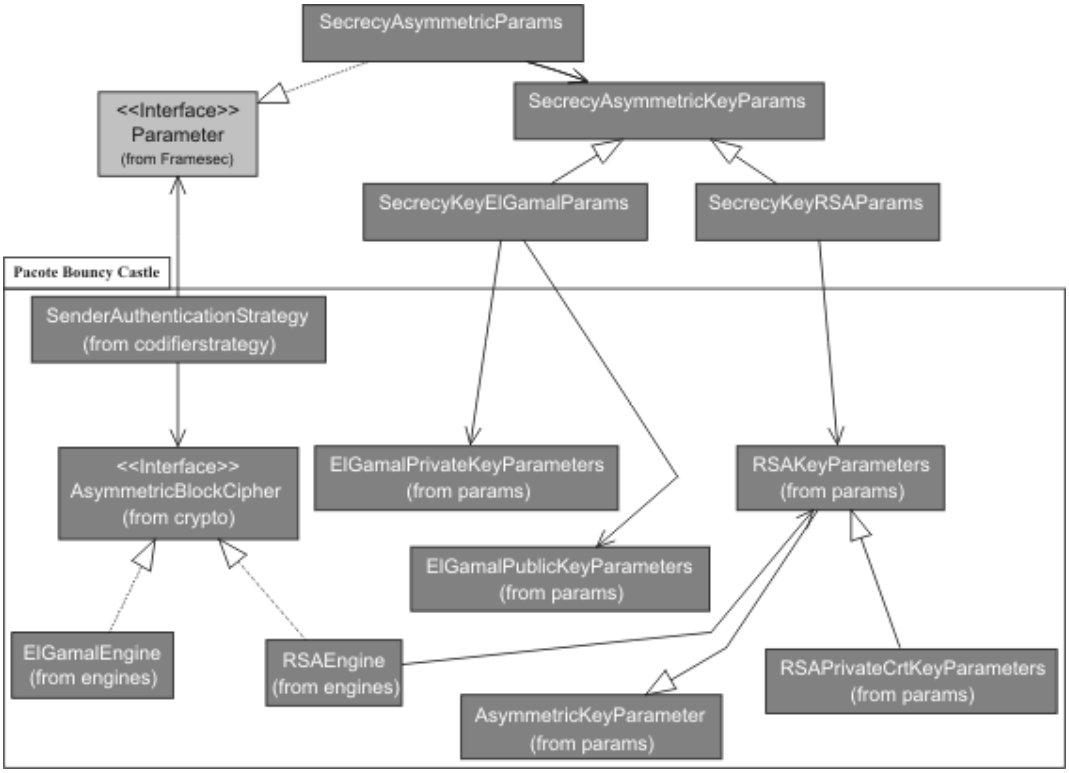

Figura 4.18. Diagrama de classes da estratégia *SenderAuthenticationStrategy*.

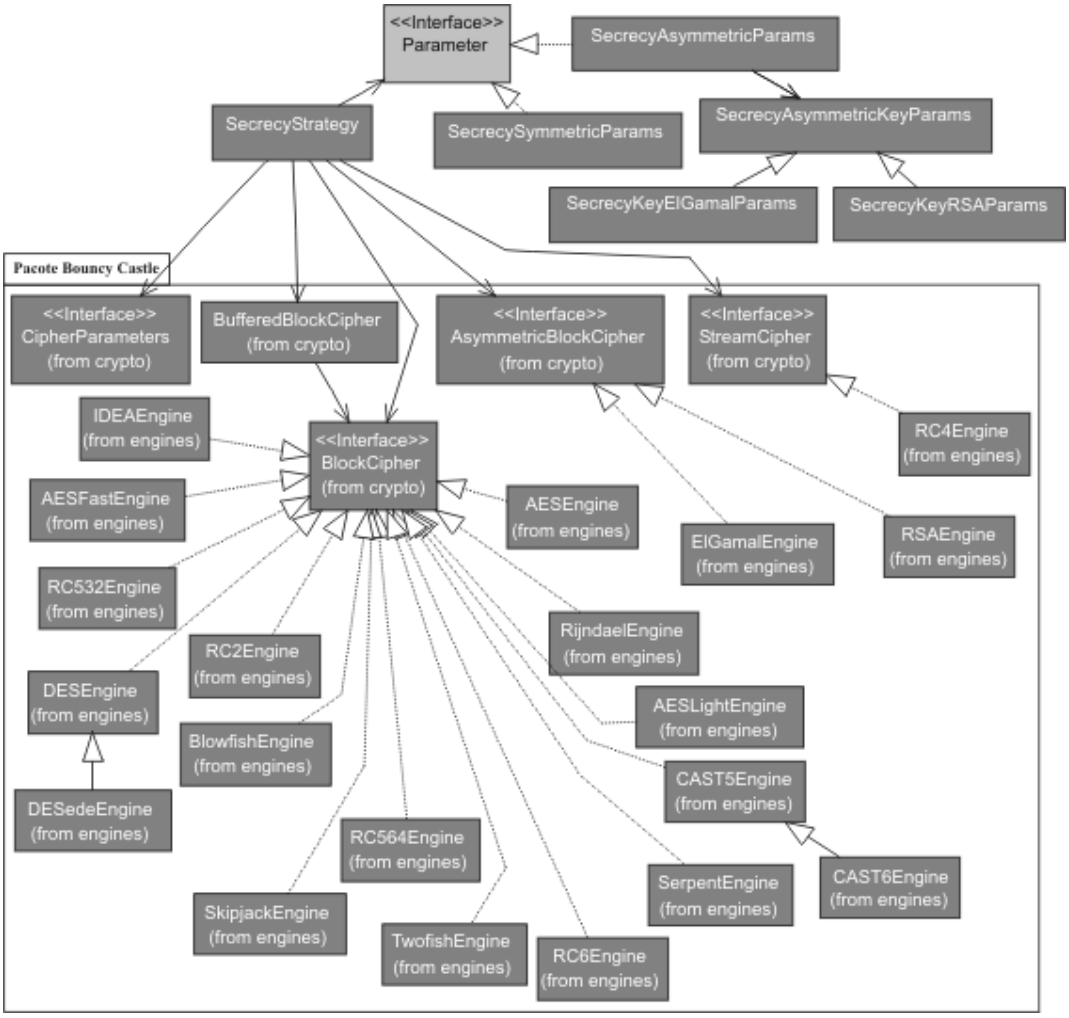

Figura 4.19. Diagrama de classes da estratégia *SecrecyStrategy*.

A estratégia MessageAuthenticationStrategy requer que o desenvolvedor informe, além do algoritmo de autenticação de mensagem a ser utilizado para garantir a autenticação da mensagem, uma função hash ou um algoritmo de criptografia simétrica, conforme apresentado na Tabela 4.5. Isto acontece porque os algoritmos de autenticação de mensagem, tais como o HMac (veja o Apêndice B), utilizam outro algoritmo de criptografia para criar o código de autenticação de mensagem, chamado de MAC (Message Authentication Code). As funções hash disponíveis para serem utilizadas em conjunto com o algoritmo de autenticação de mensagem HMac são listados na Tabela 4.2. Por sua vez, os algoritmo de criptografia simétrica disponíveis para serem utilizados em conjunto aos algoritmos de autenticação de mensagem Mac, CFBMac e CBCMac são os mesmos listados na Tabela 4.3, com exceção do algoritmo RC4, que é

um algoritmo de criptografia simétrica do tipo stream cipher (cifra de fluxo, para maiores detalhes veja o Apêndice B).

Tabela 4.3. Algoritmos de criptografia disponíveis para a *SecrecyStrategy*.

Algoritmos de criptografia simétrica e assimétrica utilizados na SecrecyStrategy	
Nome do algoritmo (Framesec)	**Classe Concreta (implementação Bouncy Castle)**
AES	AESEngine
AESFast	AESFastEngine
AESLight	AESLightEngine
Blowfish	BlowfishEngine
CAST5	CAST5Engine
CAST6	CAST6Engine
DESede	DESedeEngine
DES	DESEngine
IDEA	IDEAEngine
RC2	RC2Engine
RC4	RC4Engine (Algoritmo *stream cipher*)
RC5-32	RC532Engine
RC5-64	RC564Engine
RC6	RC6Engine
Rijndael	RijndaelEngine
Serpent	SerpentEngine
Skipjack	SkipjackEngine
Twofish	TwofishEngine
RSA	RSAEngine (Assimétrico)
ElGamal	ElGamalEngine (Assimétrico)

Tabela 4.4. Algoritmos de criptografia disponíveis para a *SenderAuthenticationStrategy*.

Algoritmos de criptografia assimétrica disponíveis na SenderAuthenticationStrategy	
Nome do algoritmo (Framesec)	**Classe Concreta (implementação Bouncy Castle)**
RSA	RSAEngine
ElGamal	ElGamalEngine

Tabela 4.5. Algoritmos de criptografia disponíveis para a *MessageIntegrityStrategy*.

Algoritmos de autenticação de mensagem (MAC) disponíveis na MessageIntegrityStrategy	
Nome do algoritmo (Framesec)	**Classe Concreta (implementação Bouncy Castle)**
HMac	HMac (requer uma função hash)
Mac	BlockCipherMac (requer um algoritmo simétrico)
CFBMac	CFBBlockCipherMac (requer um algoritmo simétrico)
CBCMac	CBCBlockCipherMac (requer um algoritmo simétrico)

Nos exemplos abaixo, são apresentados a utilização de cada estratégia na instanciação do Framesec implementado em J2ME, exceto a *IntegrityStrategy* já apresentado na Figura 4.12 e na Figura 4.13. É suposto que os algoritmos escolhidos pelo desenvolvedor, conforme os resultados das avaliações de desempenho realizadas

através da ferramenta PEARL no DM das funções *hash*, dos algoritmos de criptografia simétrica e assimétrica são, respectivamente, o MD5, o Rijndael e o RSA.

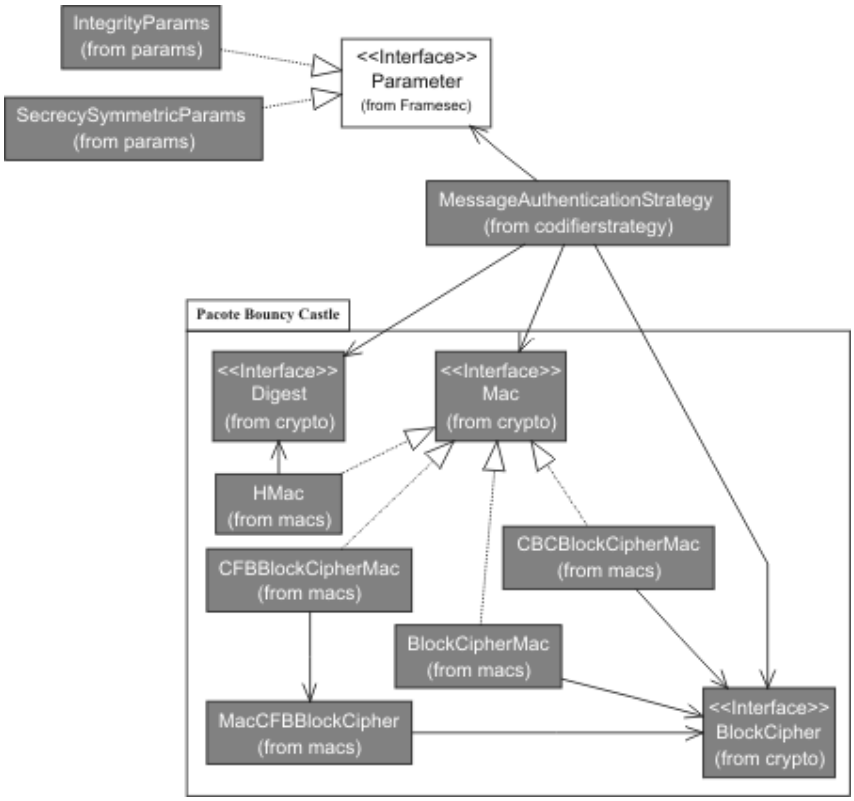

Figura 4.20. Diagrama de classes da estratégia *MessageAuthentication.*

A Figura 4.21 ilustra a instanciação do Framesec utilizando a estratégia *SecrecyStrategy*:

```
// Criando o codificador e decodificador da mensagem
Peer alice;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext, key;

codifier = new Codifier(new SecrecyStrategy("Rijndael"));
decodifier = new Decodifier(new SecrecyStrategy("Rijndael"));
...
// Em algum momento é recuperado o valor da chave (de forma segura)

codifier.initialize(new SecrecySymmetricParams(key));
decodifier.initialize(new SecrecySymmetricParams(key));
peers = new EntryHTTP();

// Alice se comunica com Bob, então devem ser informados os dados referentes à Bob.
// Este método é executado a cada novo ponto de comunicação estabelecido por Alice
peers.addEntry("Bob", "http://200.17.37.12:8080", "post", codifier, decodifier);
```

```
alice = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.21. Exemplo de instanciação do Framesec utilizando a estratégia *SecrecyStrategy*.

A Figura 4.22 ilustra a instanciação do Framesec utilizando a estratégia *SenderAuthenticationStrategy*:

```
// Criando o codificador e decodificador da mensagem
Peer alice;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext;
SecrecyKeyRSAParams keysAlice;
SecrecyKeyRSAParams keyBob;   // somente a chave pública de Bob

codifier = new Codifier(new SenderAuthenticationStrategy("RSA"));
decodifier = new Decodifier(new SenderAuthenticationStrategy("RSA"));
...
// Em algum momento são recuperadas as chaves publicas e privadas
...
codifier.initialize(new SecrecyAsymmetricParams(keysAlice, keyBob));
decodifier.initialize(new SecrecyAsymmetricParams(keysAlice, keyBob));
peers = new EntryHTTP();

// Alice se comunica com Bob, então devem ser informados os dados referentes à Bob.
// Este método é executado a cada novo ponto de comunicação estabelecido por Alice
peers.addEntry("Bob", "http://200.17.37.12:8080", "post", codifier, decodifier);
alice = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.22. Exemplo de instanciação do Framesec utilizando a estratégia *SecrecyStrategy*.

A Figura 4.23 a ilustra a instanciação do Framesec utilizando a estratégia *MessageAuthenticationStrategy*:

```
// Criando o codificador e decodificador da mensagem
Peer alice;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext, key;

codifier = new Codifier(new MessageAuthenticationStrategy(“HMac”, “MD5”));
decodifier = new Decodifier(new MessageAuthenticationStrategy(“HMac”, “MD5”));
...
// Em algum momento é recuperada a chave
...
codifier.initialize(new SecrecySymmetricParams(key));
decodifier.initialize(new SecrecySymmetricParams(key));
peers = new EntryHTTP();

// Alice se comunica com Bob, então devem ser informados os dados referentes à Bob.
// Este método é executado a cada novo ponto de comunicação estabelecido por Alice
peers.addEntry(“Bob”, “http://200.17.37.12:8080”, “post”, codifier, decodifier);
alice = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, “Bob”);
...
/* Recebendo a mensagem */
plaintext = alice.receive(“Bob”);
```

Figura 4.23. Exemplo de instanciação do Framesec utilizando a estratégia *MessageAuthenticationStrategy*.

Por fim, a Figura 4.23 a ilustra a instanciação do Framesec utilizando a estratégia *SecrecyIntegrityStrategy*:

```
// Criando o codificador e decodificador da mensagem
Peer alice;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
byte[] plaintext, ciphertext, key;

codifier = new Codifier(new SecrecyIntegrityStrategy(“Rijndael”, “MD5”));
decodifier = new Decodifier(new SecrecyIntegrityStrategy(“Rijndael”, “MD5”));
...
// Em algum momento é recuperada a chave
...
codifier.initialize(new SecrecySymmetricParams(key));
decodifier.initialize(new SecrecySymmetricParams(key));
peers = new EntryHTTP();
```

```
// Alice se comunica com Bob, então devem ser informados os dados referentes à Bob.
// Este método é executado a cada novo ponto de comunicação estabelecido por Alice
peers.addEntry("Bob", "http://200.17.37.12:8080", "post", codifier, decodifier);
alice = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.24. Exemplo de instanciação do Framesec utilizando a estratégia *SecrecyIntegrityStrategy*.

Passo 6.5: Elaboração do Roteiro de Utilização

Objetivo: Explicar, sob a ótica do desenvolvedor, como construir aplicações para DMs que possuem requisitos de integridade, confidencialidade, autenticidade do emissor e autenticidade da mensagem, a partir das instâncias do Framesec.

a) Instanciando o Framesec

Para instanciar o Framesec, o desenvolvedor deve decidir qual a estratégia que será utilizada (i.e., *IntegrityStrategy, SecrecyStrategy, SenderAuthenticationStrategy, MessageAuthenticationStrategy e SecrecyIntegrityStrategy*) e quais os algoritmos criptográficos irão compor os *hot spots* do *framework*, conforme mencionado anteriormente, dentre os presentes na Tabela 4.2, na Tabela 4.3, na Tabela 4.4 e na Tabela 4.5.

Além desta decisão, o desenvolvedor deve optar por instanciar o Framesec por completo (que fornece a comunicação interprocesso transparente) ou não, conforme citado no Passo 2.5.

b) Utilizando um novo algoritmo de criptografia simétrica ou assimétrica

Caso o desenvolvedor deseje utilizar um novo algoritmo de criptografia simétrica e assimétrica, este deve modificar a classe *SecrecyStrategy* a fim de torná-lo visível às instâncias do Framesec.

c) Utilizando um novo algoritmo de autenticação do emissor

Caso o desenvolvedor deseje utilizar um novo algoritmo de autenticação do emissor, este deve modificar a classe *SenderAuthenticaionStrategy* a fim de torná-lo visível às instâncias do Framesec.

d) Utilizando um novo algoritmo de autenticação de mensagem

Caso o desenvolvedor deseje utilizar um novo algoritmo de autenticação de mensagem, este deve modificar a classe *MessageAuthenticationStrategy* a fim de torná-lo visível às instâncias do Framesec.

Com o término deste passo o 5º ciclo de desenvolvimento do Framesec foi finalizado. O *framework* definido neste passo fornece os serviço de integridade, confidencialidade, autenticação do emissor, autenticação da mensagem e confidencialide com integridade, às aplicações construídas a partir das instância do Framesec. Será apresentado agora o décimo passo executado para a construção do Framesec, conforme citado na Seção 4.4.

Passo 10: 9º Ciclo de Desenvolvimento

Ao final deste passo teremos o 9º *framework* construído, dentre a família de *frameworks* gerada, que fornece todas as funcionalidades de segurança ilustrados na Figura 4.6. As funcionalidade de segurança adicionadas neste ciclo é fornecida através da classe *SecrecySignatureAppendixStrategy* adicionada ao diagrama de classes, como será visto mais adiante.

Passo 10.1: Coleta de Requisitos

a) Requisitos Funcionais

- Prover confidencialidade com assinatura do resumo (*Secrecy with Signature with Appendix*).

b) Requisitos Não-Funcionais

- Ser extensível, possibilitando adicionar novos algoritmos de criptografia simétrica, assimétrica e funções *hash* (veja o Apêndice B) para prover a confidencialidade com assinatura do resumo;
- Ser de fácil compreensão;
- Conter documentação.

c) Descrição dos Casos de Uso

01 – Caso de Uso Abstrato Enviar Mensagem

Descrição: é calculado o MDC da mensagem a ser enviada utilizando a função *hash* combinada entre as partes comunicantes. O MDC calculado é assinado e concatenado à mensagem a ser enviada. Por fim, a mensagem concatenada com o MDC assinado é encriptada utilizando o algoritmo de criptografia e chaves combinadas entre as partes comunicantes, sendo posteriormente enviada ao *Peer* de destino.

02 – Caso de Uso Abstrato Codificar Mensagem

Descrição: é calculado o MDC da mensagem utilizando a função *hash* desejada. O MDC calculado é assinado e concatenado à mensagem. A mensagem concatenada com o MDC assinado é encriptada utilizando o algoritmo de criptografia e as chaves desejadas, sendo posteriormente retornada ao objeto que requisitou a codificação.

03 – Caso de Uso Abstrato Receber Mensagem

Descrição: é realizada a decriptação da mensagem recebida utilizando o algoritmo de criptografia e as chaves combinadas entre as partes comunicantes. Após isso, a mensagem é separada do MDC assinado enviado. O MDC assinado é decriptado utilizando a chave pública do emissor e, a partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC enviado e o calculado, a fim de verificar se não houve modificações durante a transmissão. Se forem iguais, o recebimento foi realizado com sucesso e a integridade, bem como confidencialidade e a autenticidade do emissor, foram verificadas.

04 – Caso de Uso Abstrato Decodificar Mensagem

Descrição: é realizada a decriptação da mensagem utilizando o algoritmo de criptografia e as chaves desejadas. Após isso, a mensagem é separada do MDC assinado. O MDC assinado é decriptado utilizando a chave pública e, a partir da mensagem propriamente dita, é recalculado o MDC. Serão comparados o MDC passado e o calculado, a fim de verificar se não houve modificações. Se forem iguais, a integridade, bem como a confidencialidade e a autenticidade da mensagem, foram verificadas.

Passo 10.2: Análise e Detecção de Hot Spots

a) Análise

A classe *SecrecySignatureAppendixStrategy* é adicionada ao diagrama de classes da análise, em relação ao diagrama de classes definido no oitavo ciclo (Passo 9) de desenvolvimento do Framesec. Esta classe é responsável pelo fornecimento do serviço de segurança confidencialidade com assinatura do resumo (padrão *Secrecy with Signature with Appendix*). A Figura 4.25 ilustra o diagrama de classes definido neste passo.

b) Detecção de *Hot Spots*

Neste ciclo de desenvolvimento não foram detectados *hot spots*. Isto por que a estratégia adicionada para o provimento do serviço de confidencialidade com integridade (*SecrecyIntegrityStrategy*) utiliza duas estratégias previamente definidas, a *SecrecyStrategy* e *IntegrityStrategy*. Estas duas estratégias fornecem os *hot spots* que permitem adicionar novos algoritmos de criptografia, sendo assim, a estratégia *SecrecyIntegrityStrategy* herdará esta flexibilidade.

Passo 10.3: Projeto

a) Definição de Padrões de Projeto

Com a adição da estratégia *SecrecySignatureAppendixStrategy* não foram necessárias a utilização de novos padrões (utilizou o padrão *Strategy* já adicionado no Passo 2 do processo de desenvolvimento do Framesec).

b) Diagrama de Classes do Projeto

Este diagrama corresponde à definição da estrutura do 9º Framesec, que apresenta suporte a todos os serviços ilustrados na Figura 4.6. A Figura 4.26 ilustra o diagrama de classes completo deste ciclo. Nesta figura, as classes do pacote *Bouncy Castle* foram suprimidas com o intuito de manter a legibilidade do diagrama.

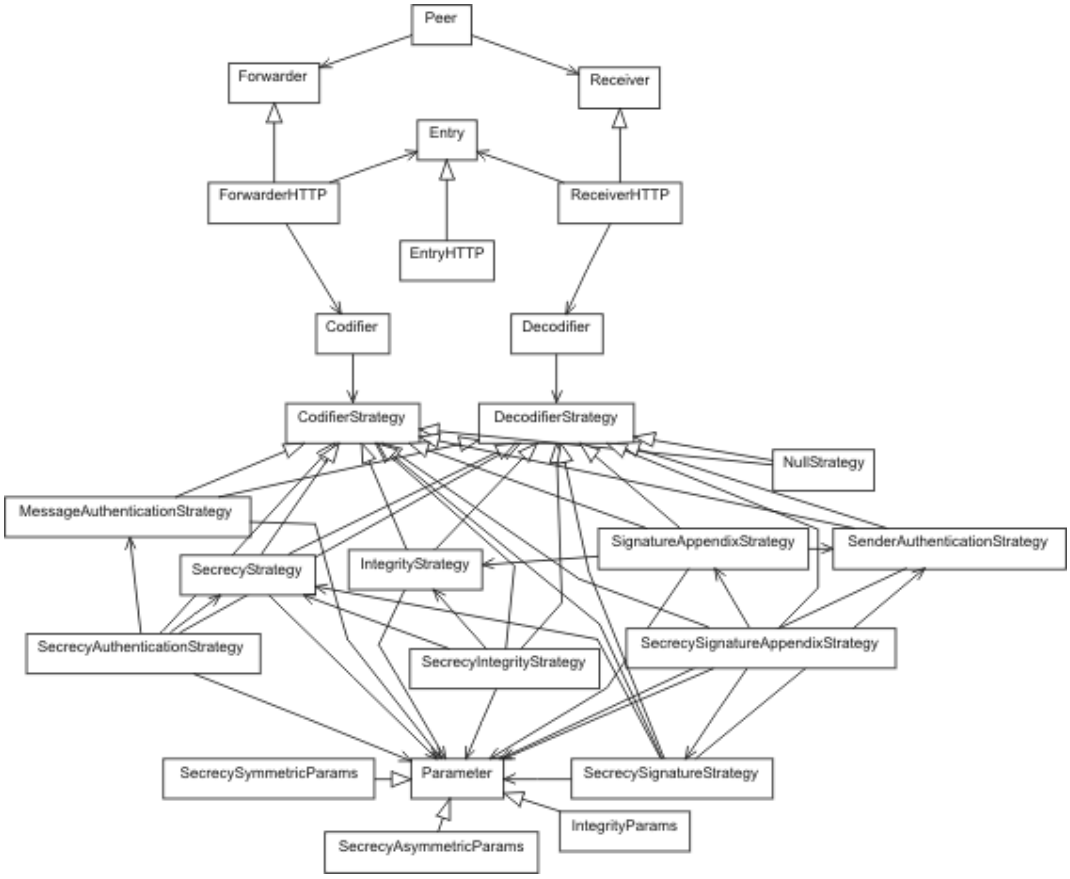

Figura 4.25. Diagrama de Classes da análise no 9º Ciclo de desenvolvimento

Passo 10.4: Implementação e instanciação Exemplo

a) Instanciação Exemplo

O processo de instanciação do *framework* é o mesmo que foi detalhado no passo 2.4 do primeiro ciclo de desenvolvimento do Framesec. No entanto, o desenvolvedor deve decidir por qual estratégia de segurança utilizar e quais os algoritmos criptográficos irão compor os *hot spots* do Framesec.

A primeira escolha a ser feita pelo desenvolvedor é a estratégia de segurança que será utilizada. De acordo com a estratégia escolhida, o desenvolvedor deverá utilizar os sub-módulos de avaliação da ferramenta PEARL, conforme descrito da Seção 5.3.2, para identificar os algoritmos que possuem o melhor desempenho no DM analisado.

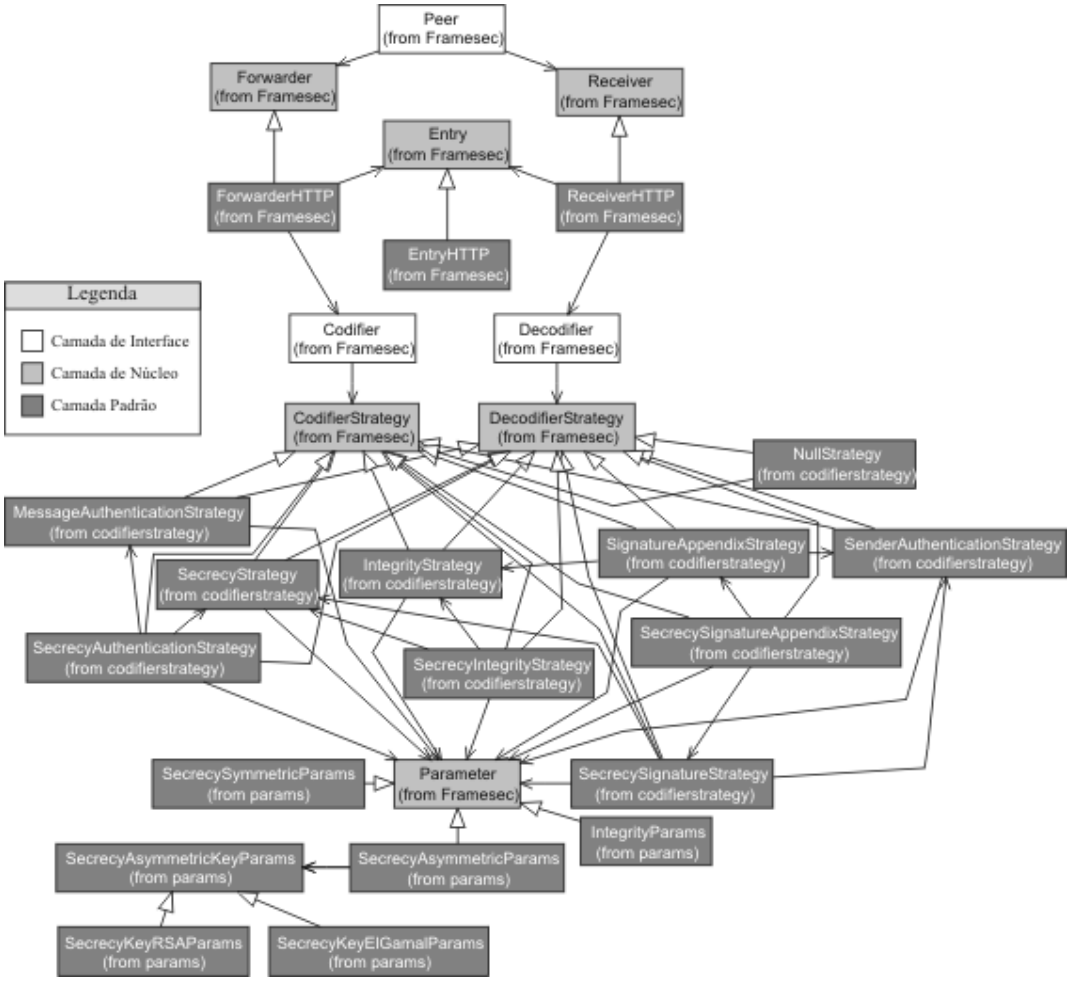

Figura 4.26. Diagrama de classe do 9º Framesec.

A estratégia *SignatureAppendixStrategy*, por combinar as estratégias *SenderAuthenticationStrategy* e *IntegrityStrategy*, deve utilizar os algoritmos disponíveis ao Framesec implementado em J2ME listados na Tabela 4.4 e na Tabela 4.2, respectivamente. Da mesma forma, a estratégia *SecrecySignatureStrategy* utiliza os algoritmos listados na Tabela 4.3 e na Tabela 4.4, a estratégia *SecrecyAuthenticationStrategy* utiliza os algoritmos listados na Tabela 4.3 e na Tabela 4.5 e, por fim, a estratégia *SecrecySignatureAppendix* utiliza os algoritmos listados na Tabela 4.3, na Tabela 4.4 e na Tabela 4.2.

Nos exemplos abaixo são apresentados a utilização de cada estratégia na instanciação do Framesec implementado em J2ME, exceto as já apresentadas no Passo 2 e Passo 6. É suposto que os algoritmos escolhidos pelo desenvolvedor, conforme os resultados das avaliações de desempenho realizadas através da ferramenta PEARL no

DM, das funções *hash*, dos algoritmos de criptografia simétrica e assimétrica são, respectivamente, o MD5, o Rijndael e o RSA.

A Figura 4.27 ilustra uma instanciação exemplo do Framesec utilizando a estratégia *SignatureAppendixStrategy*:

```
// Criando o codificador e decodificador da mensagem
...
codifier = new Codifier(new SignatureAppendixStrategy("RSA", "MD5"));
decodifier = new Decodifier(new SignatureAppendixStrategy("RSA", "MD5"));
...
// Em algum momento é recuperado o valor da chave (de forma segura)
...
codifier.initialize(new SecrecySymmetricParams(key));
decodifier.initialize(new SecrecySymmetricParams(key));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.27. Instanciação exemplo do Framesec utilizando a estratégia *SignatureAppendixStrategy*.

A Figura 4.28 ilustra uma instanciação exemplo do Framesec utilizando a estratégia *SecrecySignatureStrategy*:

```
// Criando o codificador e decodificador da mensagem
...
codifier = new Codifier(new SecrecySignatureStrategy("Rijndael", "RSA"));
decodifier = new Decodifier(new SecrecySignatureStrategy("Rijndael", "RSA"));
...
// Em algum momento são recuperadas as chaves publicas e privadas
...
codifier.initialize(new SecrecySymmetric(key), new
SecrecyAsymmetricParams(keysAlice, keyBob));
decodifier.initialize(new SecrecySymmetric(key), new
SecrecyAsymmetricParams(keysAlice, keyBob));
...
/* o plaintext possui o valor da mensagem a ser codificada e enviada. */
alice.send(plaintext, "Bob");
...
/* Recebendo a mensagem */
plaintext = alice.receive("bob");
```

Figura 4.28. Instanciação exemplo do Framesec utilizando a estratégia *SecrecySignatureStrategy*.

A Figura 4.29 a ilustra uma instanciação exemplo do Framesec utilizando a estratégia *SecrecyAuthenticationStrategy*:

```
// Criando o codificador e decodificador da mensagem
...
codifier = new Codifier(new SecrecyAuthenticationStrategy("HMac", "Rijndael"));
decodifier = new Decodifier(new SecrecyAuthenticationStrategy("HMac",
"Rijndael"));
...
codifier.initialize(new SecrecySymmetricParams(key));
decodifier.initialize(new SecrecySymmetricParams(key));
...
// Processo idêntico ao dos demais exemplos
```

Figura 4.29. instanciação exemplo do Framesec utilizando a estratégia *SecrecyAuthenticationStrategy*.

A Figura 4.30 a ilustra uma instanciação exemplo do Framesec utilizando a estratégia *SecrecySignatureAppendixStrategy*:

```
// Criando o codificador e decodificador da mensagem
...
codifier = new Codifier(new SecrecySignatureAppendixStrategy("Rijndael", "RSA",
"MD5"));
decodifier = new Decodifier(new SecrecySignatureAppendixStrategy("Rijndael",
"RSA", "MD5"));
...
codifier.initialize(new SecrecySymmetric(key), new
SecrecyAsymmetricParams(keysAlice, keyBob));
decodifier.initialize(new SecrecySymmetric(key), new
SecrecyAsymmetricParams(keysAlice, keyBob));
...
// Processo idêntico ao dos demais exemplos
```

Figura 4.30. Framesec utilizando a estratégia *SecrecySignatureAppendixStrategy*.

Objetivo: Explicar, sob a ótica do desenvolvedor, como construir aplicações para DMs que possuem requisitos de integridade, confidencialidade, autenticidade do emissor e autenticidade da mensagem, a partir das instâncias do Framesec.

a) Instanciando o Framesec

Para instanciar o Framesec, o desenvolvedor deve decidir qual a estratégia que será utilizada e quais os algoritmos criptográficos irão compor os *hot spots* do

framework, conforme mencionado anteriormente, dentre os presentes na Tabela 4.2, na Tabela 4.3, na Tabela 4.4 e na Tabela 4.5.

Além desta decisão, o desenvolvedor deve optar por instanciar o Framesec por completo (que fornece a comunicação interprocesso transparente) ou não, conforme citado no Passo 2.5.

Com o término deste passo o 9º ciclo de desenvolvimento do Framesec foi finalizado. O *framework* definido neste passo fornece às aplicações construídas a partir das instância do Framesec todos os serviço de seguraça ilustrados na Figura 4.6. Será apresentado agora o décimo primeiro passo executado para a construção do Framesec, conforme citado na Seção 4.4.

Passo 11: Instanciando Aplicações

Ao término do décimo passo a família de *framework* de segurança foi completada. O último passo correspondente a instanciação de aplicações será demonstrada no Capítulo 6, onde é apresentada a aplicação *MobileMulta* construída como estudo de caso deste trabalho. Esta aplicação utiliza os serviços de segurança fornecidos pelo *framework*, especificamente utilizando a estratégia *SecrecyIntegrityStrategy*.

A próxima seção apresenta a conclusão deste capítulo.

4.5 Conclusão

O Framesec apresentado neste capítulo permite a construção de mecanismos de segurança fim-a-fim, a serem adicionados às aplicações no ambiente de computação móvel considerado neste trabalho.

Foram apresentados três ciclos de desenvolvimentos executados para a construção do Framesec, bem como instâncias exemplos de utilização de cada serviço de segurança disponibilizado pelo *framework*, através da utilização das estratégias de segurança e dos algoritmos criptográficos do pacote *Bouncy Castle*, implementados na plataforma J2ME. Nas instâncias exemplos, os algoritmos foram escolhidos pelo desenvolvedor supondo os resultados das avaliações de desempenho dos algoritmos criptográficos nos DMs.

No Passo 1 de construção do Framesec, foi definido que o *framework* não fornece os mecanismos de troca de chave, bem como o mecanismo que deve ser executado previamente para a combinação da estratégia e dos algoritmos a serem

utilizados pelas partes comunicantes (i.e., DM e o SA) para a provisão da segurança fim-a-fim. A adição desses serviços ao Framesec será realizada em trabalhos futuros.

Por fim, a ferramenta PEARL proposta neste trabalho, que foi utilizada pelo desenvolvedor para avaliar os algoritmos criptográficos nos DM de forma a permitir escolher os algoritmos que seriam utilizados nas instâncias exemplos apresentadas neste capítulo, será apresentada em detalhes no capítulo seguinte.

5 Uma Ferramenta para Avaliação de Desempenho de Algoritmos Criptográficos em Dispositivos Móveis

Conforme citado no Capítulo 4, alguns fatores podem influenciar na escolha dos algoritmos que irão compor os *hot spots* da instância do Framesec, tais como desempenho e segurança provida pelo algoritmo. O desempenho pode atuar como fator limitante da utilização de alguns algoritmos em um determinado DM que apresenta limitação de recursos. O fator segurança não será abordado neste trabalho, ficando o mesmo para trabalhos futuros. Sendo assim, é necessário realizar as avaliações de desempenho dos algoritmos criptográficos que irão compor os *hot spots* do Framesec no DM a qual as aplicações se destinam, antes do início do desenvolvimento de uma aplicação que possui requisitos de segurança.

Este capítulo apresenta a ferramenta PEARL (*Performance Evaluator of CriptogRaphic aLgorithms for mobile device*), que permite avaliar o desempenho de algoritmos de criptografia simétrica, assimétrica e funções hash, implementados para a plataforma J2ME. Antes de apresentar a ferramenta em detalhes, serão discutidos os trabalhos relacionados, bem com a abordagem de avaliação definida e utilizada na construção da ferramenta PEARL.

5.1 Introdução

De acordo com as pesquisas realizadas na literatura, o trabalho mais diretamente relacionado à ferramenta PEARL é descrito em [56]. No trabalho citado, os autores implementaram e avaliaram várias bibliotecas de sistemas criptográficos para o Palm OS, incluindo implementações na linguagem C de *stream ciphers* (SSC2, ARC4 e SEAL 3.0), *block ciphers* (Rijndael, DES, DESX e TripleDES), funções *hash* (MD2, MD4, MD5 e SHA-1) e operações aritméticas inteiras de múltipla precisão, que são essenciais para a implementação de algoritmos de criptografia assimétrica.

Os algoritmos foram avaliados, em termos de desempenho, nos dispositivos Palm V (Processador 16 Mhz, 2Mb) e Palm IIIc (Processador 20 Mhz, 8Mb). As avaliações consistiam na execução dos algoritmos nesses dispositivos, encriptando *plaintext* de tamanhos variados (2 Kb, 50 Kb e 4Mb) onde eram calculadas as

quantidades de bytes encriptados (ou resumidos, no caso de funções *hash*) a cada segundo (bytes/s).

Com o resultado das avaliações, os autores identificaram que o *stream cipher* SSC2 obteve desempenho melhor para *plaintext* pequenos (1Kb) nos dispositivos analisados, quando comparado com o ARC4 e SEAL 3.0. Para *plaintext* grandes (4Mb) o SEAL 3.0 obteve o dobro da velocidade do SSC2. Analisando os resultados dos *block ciphers*, os autores observaram que o Rijndael é quatro vezes mais rápido que o DES.

Além disso, os autores demonstraram a viabilidade da utilização de operações aritméticas inteiras de múltipla precisão nos dispositivos analisados. Por fim, foi observado pelos autores que implementações dos algoritmos criptográficos embutidos nas aplicações possuem melhor desempenho que as implementações baseadas em bibliotecas de sistemas.

Vale ainda ressaltar o trabalho descrito em [57], onde os autores realizaram uma análise de desempenho de protocolos de segurança nos dispositivos móveis, tais como o SSL, S/MIME e IP/Sec, que são largamente utilizados em aplicações de redes. Os resultados da análise demonstram que as operações criptográficas executadas pelos protocolos de segurança SSL, S/MIME e IP/Sec, não degradam de forma significante o desempenho de transações *real-time* envolvendo dispositivos móveis, tais como PDAs e telefones móveis. Entretanto, a análise foi realizada apenas no dispositivo iPAC H3630 (processador StrongARM de 206 Mhz, 32 Mb de RAM com o sistema operacional Windows CE Pocket PC 2002). Este dispositivo possui desempenho similar a de um computador *desktop*, o que deixa em aberto a validade da análise para dispositivos móveis de menor poder computacional.

No entanto, a ferramenta PEARL proposta, em comparação com os trabalhos anteriores, permite avaliar o desempenho dos algoritmos criptográficos em qualquer dispositivo móvel, desde que este apresente suporte à plataforma J2ME/MIDP 1.0. O desempenho de cada algoritmo é avaliado de acordo com a quantidade de tempo necessário para executar operações criptográficas sobre uma entrada de texto de tamanho definido. Os resultados das avaliações são apresentados na forma de *throughput* (bytes/s), que corresponde à quantidade de bytes que podem ser encriptados ou resumidos por segundo.

5.2 Abordagem utilizada para a Avaliação de Desempenho de Algoritmos Criptográficos em Dispositivos Móveis

A abordagem para a avaliação de desempenho utilizada e implementada na ferramenta PEARL se estrutura em duas partes explicadas a seguir. A primeira, mais quantitativa, permite coletar e agrupar informações referentes às execuções dos algoritmos nos dispositivos móveis. A segunda parte se propõe a avaliar essas mesmas informações e identificar, a partir dos resultados desta avaliação, quais algoritmos possuem melhor desempenho no dispositivo analisado.

Antes de apresentar os detalhes da ferramenta PEARL, serão descritos o formato da amostra, que consiste no agrupamento de informações resultantes de uma execução de um algoritmo em um dado dispositivo, o processo de coleta, e o processo de análise da abordagem utilizada.

5.2.1 Descrição do formato da amostra

As amostras contêm informações referentes às execuções dos algoritmos criptográficos no dispositivo analisado. O formato da amostra permite identificar o DM analisado, a máquina virtual Java corrente (o DM pode apresentar suporte a várias máquinas virtuais diferentes), o algoritmo avaliado, o tamanho da mensagem encriptada ou resumida, e tempos de inicialização e execução do algoritmo.

Essas informações são suficientes para avaliar o desempenho dos algoritmos criptográficos em cada DM, para uma dada máquina virtual Java. A cada execução de um algoritmo com um tamanho de entrada pré-definido, uma amostra é criada e armazenada para futura análise. O formato da amostra contém os seguintes campos:

- *idDevice*: número inteiro que identifica o dispositivo analisado. Cada dispositivo, antes de iniciar a avaliação, recebe um identificador único;
- *idVM:* número inteiro que identifica unicamente a máquina virtual Java utilizada para avaliar os algoritmos criptográficos. Este campo é opcional, já que a abordagem pode ser implementada em outra linguagem (e.g., C);
- *idAlg:* número inteiro que identifica unicamente o algoritmo avaliado;
- *sizeText:* tamanho do texto de entrada, pois o mesmo algoritmo é avaliado para tamanho de textos variados;

– *timeInit:* tempo de inicialização do algoritmo, quando for possível coletar, que pode variar conforme o tamanho da entrada;
– *timeExec:* tempo de execução do algoritmo. Para cada tamanho de entrada e algoritmo avaliado teremos um tempo de execução diferente.

5.2.2 Coleta das amostras

A amostra coletada a cada execução de um algoritmo com uma entrada de tamanho específico, é armazenada em um repositório de amostras local no DM (observe na Figura 5.2, o *Samples Local Base* – SLB).

Um *ensaio* corresponde a uma execução de um algoritmo com um tamanho de entrada específico, que gera como resultado uma amostra. Para cada par tamanho de entrada e algoritmo, vários ensaios são realizados, cada um gerando uma amostra para futura análise. A quantidade de ensaios realizados para cada tamanho de entrada e algoritmo é definida na implementação, no caso da ferramenta PEARL, o usuário pode escolher este valor.

Visando obter maior precisão na avaliação de desempenho dos algoritmos, estes são executados sobre textos de tamanho variados de entrada. Este tamanho cresce de forma linear, permitindo observar as variações nos tempos de execução e inicialização dos algoritmos, conforme o aumento do tamanho da entrada.

Os tamanhos iniciais e finais dos textos de entrada utilizados como padrão são, respectivamente, 1Kb e o tamanho máximo múltiplo de dois de um vetor de bytes possível de ser alocado por uma aplicação hospedada no DM. No caso da ferramenta PEARL, que é implementada na plataforma J2ME, este tramanho corresponde ao tamanho máximo de um vetor de bytes que a máquina virtual utilizada consegue alocar. O tamanho inicial é duplicado até atingir o tamanho máximo definido por padrão, ou informado pelo usuário da ferramenta que implementa esta abordagem. Por exemplo, na ferramenta PEARL, é possível escolher os tamanhos iniciais e finais do texto a ser avaliado.

A Figura 5.1 apresenta o pseudocódigo, em Java (J2ME), da operação de coleta das amostras. A função *expTwo(k)* retorna o valor da exponencial de k na base 2 (i.e., 2^k). O primeiro laço *for* é executado até o tamanho do texto atingir o tamanho máximo (*sizeMax*). A variável *sizeplaintext* recebe o tamanho da entrada em bytes, que será encriptado ou resumido. O laço *for* mais interno representa a quantidade de ensaios

que serão realizados para o tamanho de entrada corrente, definido através da variável *sizeplaintext*. A cada ensaio são calculados os tempos de inicialização e execução do algoritmo avaliado, para o tamanho de entrada corrente.

```
for (int k = 0; expTwo(k) <= sizeMax; k++){
   sizeplaintext = expTwo(k) * 1024;
   for (int j = 0; j < amountExperiment; j++){
      // Iniciando o algoritmo
      StartTimeInit();
      Cipher.init(params);
      StopTimeInit();

       // Executando a cifragem
      StartTimeCipher();
      Cipher.Execute(plaintext,sizeplaintext);
      StopTimeCipher();

      // Armazenando a amostra
      Sample.persist();
   }
}
```

Figura 5.1. Pseudocódigo da operação de coleta de amostras.

Para realizar os ensaios com as funções *hash*, o pseudocódigo ilustrado na Figura 5.1 sofre algumas modificações, já que eles executam diretamente a função sobre o *plaintext* de entrada, não sendo calculado o tempo de inicialização do algoritmo.

Após a fase de coleta, as amostras armazenadas no repositório de amostras local no DM são enviadas para um computador *desktop*, que possui maior poder de processamento, para serem analisadas. As amostras recebidas são armazenadas no repositório remoto de amostras (na Figura 5.2, corresponde ao *Samples Remote Base* – SRB). A análise das amostras é realizada, portanto, sobre as amostras no repositório remoto localizado no computador *desktop*, e não no dispositivo móvel.

5.2.3 Análise das amostras

Nesta abordagem, a análise das amostras de um algoritmo tem como resultado a quantidade de bytes encriptados ou resumidos por segundo, para cada tamanho de *plaintext*. A este valor dá-se o nome de *througthput* (bytes/s) do algoritmo.

As operações de análise são realizadas em um computador *desktop*, que são executadas sobre as informações das amostras armazenadas no repositório remoto de amostras. O por que destas operações não serem realizadas nos próprios DMs, decorre

do fato de que estes apresentam limitações de recursos, o que tornaria lenta a execução destas operações quando as mesmas fossem possíveis de serem implementadas no DM.

Para se calcular o *througthput* dos algoritmos criptográficos, são utilizados os tamanhos do texto de entrada (e.g., 1024 Bytes, 2048 Bytes, 4096 Bytes) e o valor modal do tempo de execução para aquele tamanho. Conforme citado na Seção 5.2.2, para cada par de tamanho de entrada e algoritmo, vários ensaios são realizados.

Sendo assim, para realizar o cálculo do *througthput* (bytes/s) dos algoritmos é necessário utilizar um valor que represente de forma aproximada a distribuição dos valores dos tempos de execução e inicialização entre os ensaios realizados para aquele tamanho. A moda é uma medida estatística de localização que corresponde ao valor que mais se repete dentro de uma distribuição. Por ser uma medida rápida e representar a aproximação do valor mais típico da distribuição, esta foi escolhida para calcular o tempo de execução e inicialização do algoritmo, para um dado tamanho de entrada.

Por fim, os resultados gerados correspondentes aos *throughtputs* dos algoritmos para os tamanhos avaliados são armazenados no repositório de resultados (na Figura 5.2, corresponde ao *Result Base – RB*) localizado no computador *desktop*.

5.3 A ferramenta PEARL

A ferramenta PEARL implementa a abordagem descrita na Seção 5.2, e possui as seguintes funcionalidades:

- permite avaliar o desempenho de algoritmos criptográficos nos DMs em uma determinada máquina virtual Java;
- permite verificar a viabilidade da utilização dos algoritmos criptográficos no DM analisado;
- Permite identificar o algoritmo que possui o melhor desempenho para o par tamanho de entrada e DM;
- a utilização da ferramenta combinada ao Framesec permite instanciá-lo de forma otimizada ao DM, em termo de desempenho, para compor as aplicações;

Esta última funcionalidade é a principal justificativa para o desenvolvimento da ferramenta. Através dos resultados das avaliações de desempenho dos algoritmos no DM, o desenvolvedor de aplicações poderá escolher os algoritmos que irão compor os *hot spots* do Framesec a ser instanciado para a construção de aplicações que possuem

requisitos de segurança fim-a-fim no ambiente de computação móvel considerado nesta dissertação. A próxima seção detalha a arquitetura da ferramenta.

5.3.1 Arquitetura

A Figura 5.2 ilustra a arquitetura da ferramenta PEARL, incluindo seus componentes e o relacionamento entre eles.

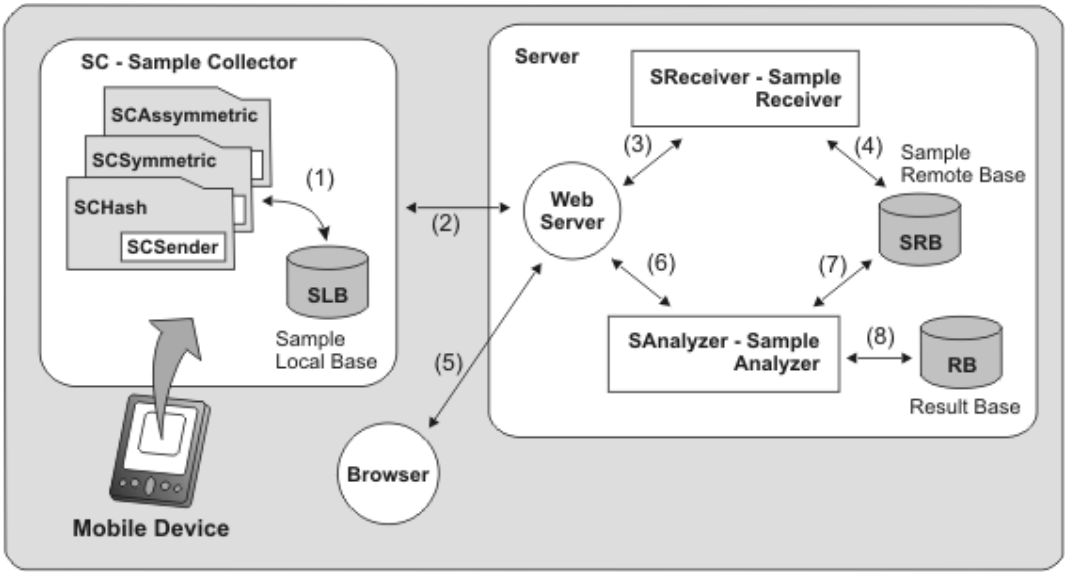

Figura 5.2. Arquitetura da PEARL: componentes e seus relacionamentos.

O módulo SC (*Sample Collector*) realiza os ensaios dos algoritmos criptográficos no dispositivo analisado. Ele é composto pelos sub-mólulos *SCAssymmetric*, *SCSymmetric* e *SCHash*, que implementam o pseudocódigo de coleta de amostras apresentado na Seção 5.2.2. Cada sub-módulo, por sua vez, possui o sub-módulo *SCSender* que é responsável por realizar o envio das amostras coletadas ao servidor para serem analisadas.

O processo de coleta das amostras é iniciado pelo usuário da ferramenta após informar os parâmetros de entrada necessários para a avaliação dos algoritmos. Neste processo, as amostras coletadas são armazenadas (1) no SLB (*Sample Local Base*).

Após o processo de coleta, o usuário utiliza o sub-módulo *SCSender* para realizar a sincronização das amostras com o servidor (2, 3). Este módulo invoca o *SReceiver* (*Sample Receiver*) para receber e armazenar as amostras enviadas (4) no SRB (*Samples Remote Base*).

Por fim, o usuário interage com o núcleo do sistema, através de um *Web Browser* (5), e invoca *SAnalyzer* (*Sample Analyzer*) para realizar a análise das amostras (6) recebidas e gera gráficos, detalhados no Capítulo 6. Este, por sua vez, recupera as amostras armazenadas (7) no SRB (*Sample Remote Base*) e efetua a sua análise, conforme descrito na Seção 5.2.3. Os resultados da análise (i.e., o *throughput* dos algoritmos para os tamanhos de entrada avaliados) são armazenados (8) no RB (*Result Base*).

O módulo SC, composto pelos sub-módulos *SCAsymmetric*, *SCSymmetric*, *SCHash* e *SCSender*, foi desenvolvido em Java na plataforma J2ME/MIDP 1.0 [71], o que garante a portabilidade do SC com uma vasta classe de DMs. O LBS foi implementado em RMS (*Record Management System*), que é o recurso de armazenamento nativo da plataforma J2ME. Por sua vez, o *SReceiver* e o *SAnalyser* foram desenvolvidos em PHP (*Hypertext Preprocessor*) [72] e hospedados no servidor Web Apache [73]. O SRB e RB foram implementados em banco de dados, utilizando o gerenciador MySQL [74].

5.3.2 SC (Sample Collector)

Conforme mencionado na Seção 5.3.1, o módulo SC é responsável por realizar os ensaios dos algoritmos criptográficos nos dispositivos móveis. Este módulo é dividido em 3 (três) sub-módulos: *SCSymmetric, SCHash* e *SCAssymetric*.

Estes sub-módulos foram desenvolvidos na plataforma J2ME, o que garante a portabilidade com os DMs que apresentam suporte ao MIDP 1.0. No processo de desenvolvimento, utilizou-se as ferramentas J2ME *Wireless Toolkit* 1.0.4_01 [71] e o SonyEricsson J2ME SDK [75]. Os algoritmos criptográficos avaliados são os mesmos utilizados para as instâncias exemplos do Framesec, conforme descrito na Seção 4.4.

Cada sub-módulo permite escolher os tamanhos inicial e final do *plaintext*, o algoritmo a ser avaliado e a quantidade de ensaios a serem realizados para cada tamanho de entrada. Além disso, cada sub-módulo também permite executar todos os algoritmos com a mesma configuração de tamanhos inicial e final do *plaintext* e a mesma quantidade de ensaios. Todos os sub-módulos utilizam o sub-módulo *SCSender* para enviar os resultados ao *SReceiver*.

Para calcular os valores *timeExec* e o *timeInit* que compõem as amostras, foi utilizada a classe *java.util.Date*, especificamente o método *getTime()*, que retorna a quantidade de milisegundos decorridos a partir das 00:00 h de 1 de janeiro de 1970.

O sub-módulo *SCSymmetric* é responsável por realizar os ensaios dos algoritmos de criptografia simétrica. Os algoritmos são divididos em 2 (duas) categorias: *block ciphers* e *stream ciphers*. Este sub-módulo permite avaliar qualquer um dos 18 (dezoito) algoritmos criptográficos, a seguir: os b*lock ciphers* AES, AES Fast, AES Light, Blowfish, CAST5, CAST6, 3DES, DES, IDEA, RC2, RC5 32 bits, RC5 64 bits, RC6, Rijndael, Serpent, Skipjack e Towfish; bem como o *stream cipher* RC4.

Alguns dos algoritmos acima são variações de algoritmos existentes, como por exemplo, o AES Fast e o AES Light, que são variações do AES. Para cada algoritmo do tipo *block chipher* são avaliados os quatro modos de encriptação. Para cada algoritmo e tamanho de entrada são realizados quatro ensaios, cada ensaio utilizando um dos modos de encriptação de bloco citado: *Electronic Codebook* (ECB), *Cipher Block Cleaning* (CBC), *Cipher Feedback* (CFB) e *Output Feedback* (OFB). Nos ensaios que utilizam os modos de encriptação CFB ou OFB, o tamanho do bloco utilizado é de 64 bits. O tamanho da chave de encriptação utilizada é de 128 bits. Para os algoritmos que utilizam chaves menores (e.g., o DES que utiliza chave de 56 Bits) ocorre o truncamento da chave para que esta passe a ter o tamanho utilizado pelo algoritmo. A Figura 5.3 ilustra os *screenshots* da execução deste sub-módulo.

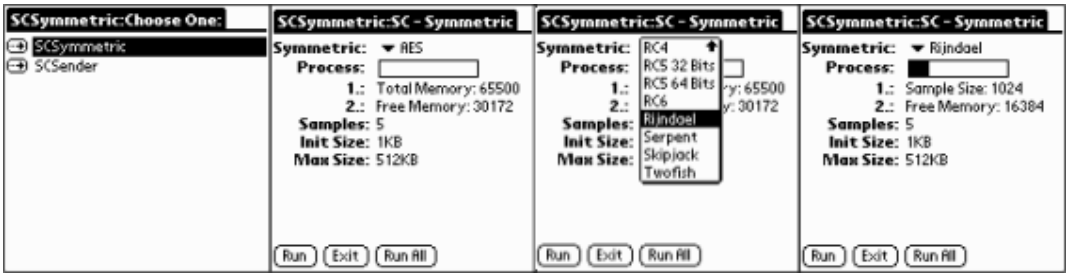

Figura 5.3. Screenshots do sub-módulo SCSymmetric.

O sub-módulo *SCHash* realiza os ensaios das funções *hash*. No total são 10 (dez) algoritmos (funções e variações): MD2, MD4, MD5, RIPEMD128, RIPEMD160, SHA 256, SHA 384, SHA 512, SHA-1 e Tiger. A Figura 5.4 ilustra os *screenshots* da execução deste sub-módulo. Conforme citado na Seção 5.2.2, este sub-módulo não realiza a coleta da informação *timeInit*, apenas o *timeExec*.

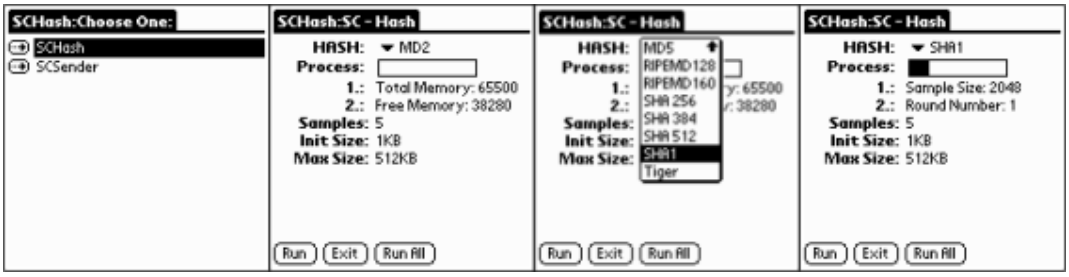

Figura 5.4. Screenshots do sub-módulo SCHash.

Por fim, o sub-módulo *SCAsymmetric* realiza os ensaios dos algoritmos de criptografia assimétrica. Este sub-módulo permite avaliar dois algoritmos de criptografia de chave pública: o RSA e o ElGamal. A chave utilizada para a avaliação destes algoritmos é de 1024 bits, por ser o tamanho mínimo de chave exigido para evitar ataques por força bruta (i.e., recuperar a chave através da geração de todas as combinações possíveis de valores). A Figura 5.5 ilustra os *screenshots* deste sub-módulo.

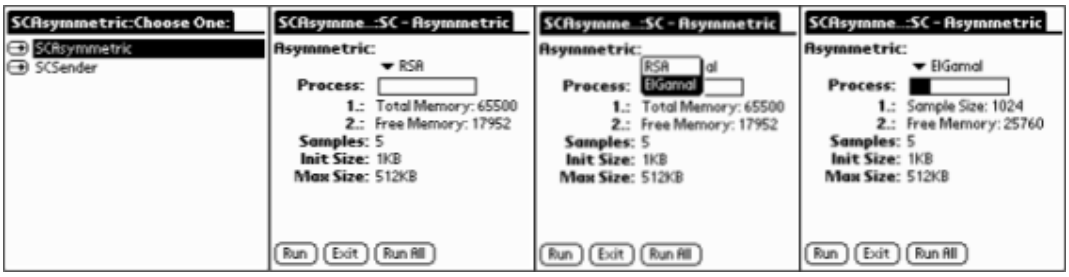

Figura 5.5. Screenshots do sub-módulo SCAsymmetric.

Os módulos SC possuem dois Casos de Uso, são eles: *Run Evaluation* e *Send Samples*. Abaixo seguem a descrição de cada caso:

Caso de Uso: *Run Evaluation*

Ator envolvido: *Developer*

Descrição: Este caso de uso se inicia no momento em que o *developer* de aplicações executa um dos sub-módulos *SCSymmetric*, *SCAsymmetric* ou *SCHash*. O *developer* deve configurar, antes de iniciar a avaliação, o algoritmo a ser executado, o número de ensaios e os tamanhos inicial e máximo do *plaintext*. Este caso de uso se encerra quando o *developer* pressiona o botão *Run* (executa o algoritmo selecionado), *Run All* (executar todos os algoritmos) ou *Exit*.

Caso de Uso: *Send Samples*

Ator envolvido: *Developer*

Descrição: Este caso de uso se inicia quando o *Developer* executa o *SCSender*. O *Developer* deve pressionar o botão *Send*. Em seguida, o *Developer* deve informar a URL contendo o endereço do SR (*Sample Receiver*) e pressionar o botão *Send*. Após isto, a ferramenta envia ao SR as amostras colhidas no caso de uso *Run Evaluator*. Este caso de uso se encerra quando o *Developer* clicar no botão *Back* (volta para a tela inicial) e, em seguida, no botão *Exit* (sai da aplicação).

5.4 Conclusão

A avaliação de desempenho dos algoritmos criptográficos é um requisito básico para a construção de mecanismos de segurança eficientes para dispositivos de baixo poder computacional. A ferramenta PEARL apresentada neste capítulo permite avaliar o desempenho dos algoritmos criptográficos em uma vasta classe de DMs (e.g., Palms, celulares, Pocket PC), o que possibilita a identificação, de forma satisfatória, dos algoritmos mais eficientes para um determinado dispositivo analisado.

Os resultados das avaliações de desempenho dos algoritmos no DM permitem ao desenvolvedor escolher, utilizando os parâmetros de desempenhos definidos pela ferramenta, os algoritmos que irão compor os *hot spots* das instâncias do Framesec implementados na plataforma J2ME. Portanto, a utilização da ferramenta combinada ao Framesec, permite instanciar os mecanismos de segurança otimizado, em termos de desempenho, ao DM analisado.

Por fim, vale ressaltar que apenas a avaliação de desempenho dos algoritmos não garante a construção de mecanismos seguros. Portanto, ainda se faz necessário um estudo das vulnerabilidades de segurança presentes nos algoritmos escolhidos pelo desenvolvedor para compor as instâncias do Framesec, proposto como trabalho futuro. O próximo capítulo apresenta o estudo de caso implementado, que utilizou a ferramenta PEARL combinada ao Framesec para a construção do mecanismo de segurança fim-a-fim, de acordo com os requisitos definidos pela aplicação.

6 Estudo de Caso

Este capítulo apresenta o estudo de caso que demonstra a utilização da ferramenta PEARL combinada ao Framesec, para a construção de mecanismo de segurança fim-a-fim para aplicação no ambiente de computação considerado neste trabalho. O capítulo foi escrito com o intuito de apresentar, sob a ótica do desenvolvedor de aplicações, a utilização da ferramenta PEARL para avaliar os algoritmos criptográficos que serão utilizados para compor os *hot spots* da instância do Framesec a ser incorporada à aplicação desenvolvida. As próximas seções, aprensentam os passos executados, em detalhes, para a construção da aplicação desenvolvida como estudo de caso.

6.1 Introdução

A aplicação *MobileMulta* desenvolvida como estudo de caso, permite ao usuário (i.e., o guarda de trânsito) realizar o cadastro de infrações de trânsito através de um DM (e.g., Palm). As infrações cadastradas no DM, que opera de forma *off-line*, são armazenadas em um repositório de dados local para posterior envio ao servidor da aplicação. Poder-se-ia realizar a transmissão das infrações no exato momento do seu cadastro, sendo necessário que o DM permaneça sempre conectado através de um dos sistema sem fio apresentados no Capítulo 2 (e.g., GPRS).

A aplicação *MobileMulta* poderá ser utilizada em qualquer um dos DMs relacionados na Tabela 6.1. A transmissão das infrações ao servidor da aplicação (i.e., a central de trânsito) a partir desses DMs, é realizada conforme a seguir: no caso dos dispositivos Palms, é estabelecida uma conexão IrDA (*Infrared Data Association*) ou *Bluetooth* com um telefone móvel que utiliza o sistema GSM/GPRS para o estabelecimento de uma conexão de dados com o SA; no caso do *smartphone* Sony Ericsson P800, é estabelecida uma conexão de dados diretamente com a rede da operadora, que utiliza os sistema GSM/GPRS. A Figura 6.1 ilustra os cenários de utilização da aplicação *MobileMulta*.

Tabela 6.1. Características dos DMs utilizados na aplicação MobileMulta.

Device	RAM	Processor	SO	VM
Palm M130	8Mb	Motorola Dragonball VZ 33 Mhz	Palm OS 4.1	IBM
Palm M515	16Mb	Motorola DragonBall VZ 33 Mhz	Palm OS 4.1	IBM
Sony Ericsson P800	12Mb	32-bit RISC ARM9 @ 156 MHz	Symbian OS 7.0	Sony Ericsson

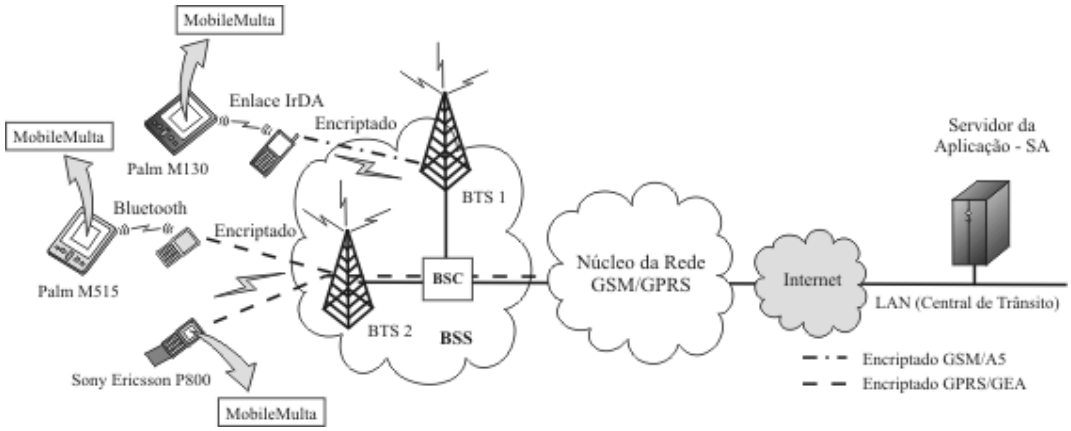

Figura 6.1. Cenário de utilização da aplicação MobileMulta.

A aplicação *MobileMulta* possui os seguintes requisitos de segurança: confidencialidade e integridade de dados. Conforme citado na Seção 4.4, a confidencialidade é o escopo do padrão *Information Secrecy* da linguagem de padrões Tropyc, que está definido no Framesec através da classe *SecrecyStrategy*. Por sua vez, a integridade é o escopo do padrão *Message Integrity*, definido no Framesec através da classe *IntegrityStrategy*.

A aplicação *MobileMulta* possui mais de um requisito de segurança, portanto, deve-se utilizar o Framesec, dentre os pertencentes à família de *frameworks*, que fornece suporte a ambos os mecanismos de confidencialidade e integridade de dados. Sendo assim, o Framesec que será utilizado é o definido no 5º Ciclo de desenvolvimento do *framework* (observe a Figura 4.6), que fornece suporte à confidencialidade e integridade através da classe *SecrecyIntegrityStrategy*. Esta estratégia utiliza os algoritmos de criptografia simétrica para prover a confidencialidade, e as funções hash para garantir a integridade dos dados.

O desenvolvedor da aplicação *MobileMulta* irá então instanciar o Framesec para a construção do mecanismo de segurança a ser incorporado à aplicação. Para tanto, este necessita definir os pontos de flexibilização da instância do *framework*, que corresponde ao algoritmo de criptografia simétrica e à função hash a serem utilizados pelos mecanismos de segurança. Portanto, será necessário avaliar o desempenho dos

algoritmos de criptografia simétrica e funções hash nos DMs descritos na Tabela 6.1, a fim de identificar os algoritmos que possuem o melhor desempenho em cada dispositivo analisado. A próxima seção descreve os detalhes da avaliação de desempenho realizada nos DMs, bem como apresenta os resultados alcançados.

6.2 Avaliação de desempenho dos Algoritmos Criptográficos nos DMs

Em cada DM da Tabela 6.1, foram avaliados o desempenho de todos os algoritmos de criptografia simétrica e funções hash. Para isso, foram instalados os sub-módulos *SCSymmetric* e *SCHash* da ferramenta PEARL nesses DMs. Conforme citado na Seção 5.2.2, os tamanhos inicial e final do *plaintext* utilizados por padrão pela ferramenta são, respectivamente, 1Kb e 256Kb.

Entretanto, não será necessária a avaliação dos algoritmos utilizando todos os tamanhos possíveis de *plaintext* múltiplos de dois, compreendidos entre 1Kb e 256Kb. Isto porque cada infração armazenada no repositório local ocupa somente 256 Bytes em memória, se o desenvolvedor levar em consideração que um guarda de transito realiza no máximo 100 infrações por dia e, ao final de cada dia de trabalho, este realizada a transmissão das infrações à central de transito, então o máximo de informações transmitidas das infrações será de 256000 Bytes, o que equivale a aproximadamente 25 Kb.

Portanto, serão utilizados os valores 1Kb e 32Kb (que é o valor ligeiramente maior que 25 múltiplo de dois) que correspondem aos tamanhos inicial e final do *plaintext* avaliado, respectivamente. Em cada dispositivo foram realizados 50 ensaios para cada tamanho de *plaintext* e algoritmo avaliado. Isto equivale a 20400 execuções de algoritmos de criptografia simétrica do tipo *block cipher*, 300 execuções do algoritmo *stream cipher* RC4 e 3000 execuções de funções hash, em cada DM analisado. Após a finalização das avaliações através dos sub-módulos *SCSymmetric* e *SCHash*, as amostras foram transmitidas ao computador *desktop* para serem analisadas pelo *Samples Analyser*, conforme o funcionamento da ferramenta descrito na Seção 5.3.1. A seguir, serão apresentados os resultados desta análise.

6.2.1 Resultados da análise

De acordo com os resultados da análise dos algoritmos de criptografia simétrica armazenados no RB (veja a Figura 5.2), pode-se concluir que em todos os DMs listados na Tabela 6.1, o algoritmo *stream cipher* RC4 apresentou o melhor desempenho. Em contrapartida, o algoritmo Rijndael apresentou o pior desempenho, não importando o modo de operação (ECB, CBC, CFB e OFB) e tamanho do *plaintext* avaliado.

A Tabela 6.2 apresenta um resumo dos resultados das avaliações dos algoritmos de criptografia simétrica, considerando o modo de operação ECB em cada DM analisado. Este modo foi utilizado na comparação, por ser o mais rápido dentre todos os quatro modos possíveis.

Tabela 6.2. Resumo das Avaliações dos Algoritmos de Criptografia Simétrica.

Device	Minor throughput (Bytes/s)	Major throughput (Bytes/s)	Throughput increasing order
Palm M130	61,02	1587,6	Rijndael < 3DES < Serpent < Skipjack < RC2 < AES Light < DES < IDEA < CAST6 < RC6 < AES < CAST5 < Twofish < AES Fast < RC5 32 Bits < BlowFish < RC5 64 Bits < RC4
Palm M515	60,97	1575,4	Rijndael < 3DES < Serpent < Skipjack < RC2 < AES Light < DES < IDEA < CAST6 <RC6 < AES < CAST5 < Twofish < AES Fast < RC5 32 Bits <BlowFish < RC5 64 Bits < RC4
SonyEricsson P800	5953,49	2048000	Rijndael < 3DES < Serpent < CAST6 < AES < AES Light < Skipjack < RC2 < IDEA < CAST5 < RC6 < DES < Twofish < RC5 32 Bits < BlowFish < AES Fast < RC5 64 Bits < RC4

Na Tabela 6.2, a coluna *Minor throughput* apresenta o valor do *throughput* do algoritmo que apresentou o pior desempenho (e.g., o valor 61,02 Bytes/s é o *throughput* do algoritmo Rijndael no dispositivo Palm M130). Seguindo o mesmo raciocínio, a coluna *Major Througput* aprensenta o valor do *througput* do algoritmo que apresentou o melhor desempenho no DM analisado. Por último, a coluna *Throughput increasing order* apresenta os algoritmos de criptografia simétrica ordenados de acordo com a ordem de crescimento do *throughput* dos algoritmos, para o *plaintext* igual a 32 Kb (i.e., tamanho final do *plaintext*).

A ferramenta PEARL também permite a visualização gráfica dos resultados da análise, o que facilita a realização de comparações gráficas dos desempenhos dos algoritmos criptográficos. A Figura 6.2, a Figura 6.3 e a Figura 6.4 ilustram os gráficos

comparativos dos quatro algoritmos de criptografia simétrica que possuem os melhores desempenhos nos dispositivos Palm M130 (Figura 6.2), Palm M515 (Figura 6.3) e Sony Ericsson P800 (Figura 6.4), respectivamente.

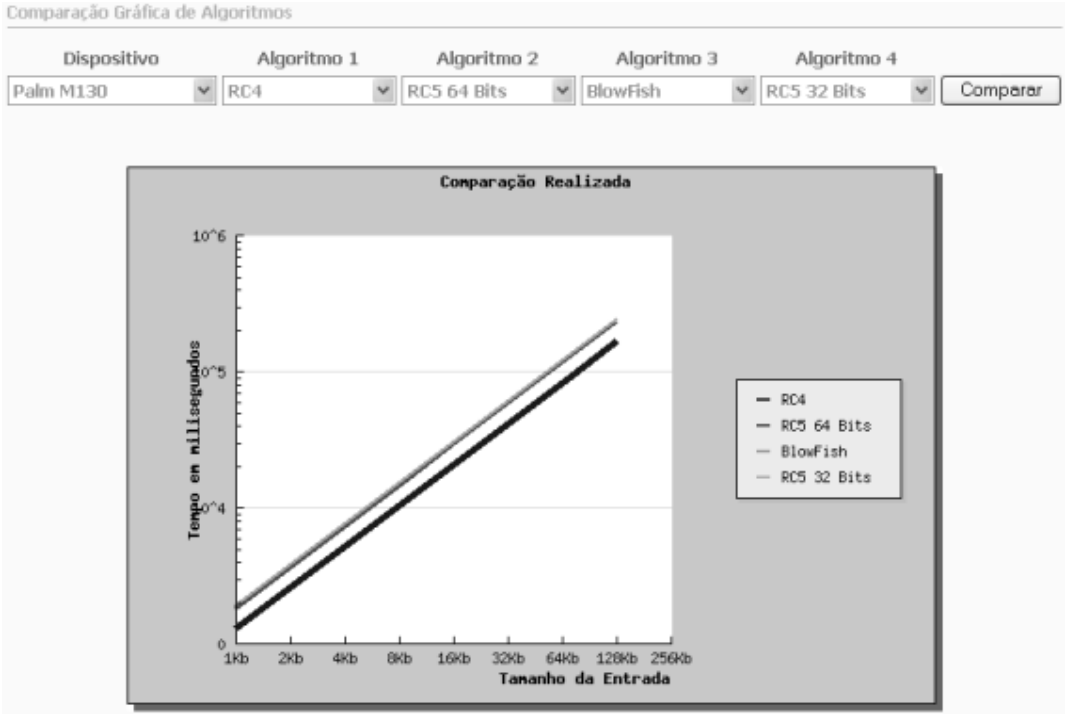

Figura 6.2. Gráfico comparativo dos quatro algoritmos de criptografia simétrica de melhor desempenho no DM Palm130.

Observando os resultados das avaliações das funções *hash*, tem-se que o algoritmo MD4 apresenta o melhor desempenho em todos os DMs analisados, não importando o tamanho do *plaintext* avaliado. Por outro lado, o MD2 apresenta o pior desempenho nos DMs, dentre todas as funções hash avaliadas. A FIGURA 6.4 apresenta um resumo dos resultados das avaliações em cada DM analisado. A ordem de crescimento de *throughput* das funções hash foi estabelecida para o *plaintext* de tamanho igual a 32 Kb.

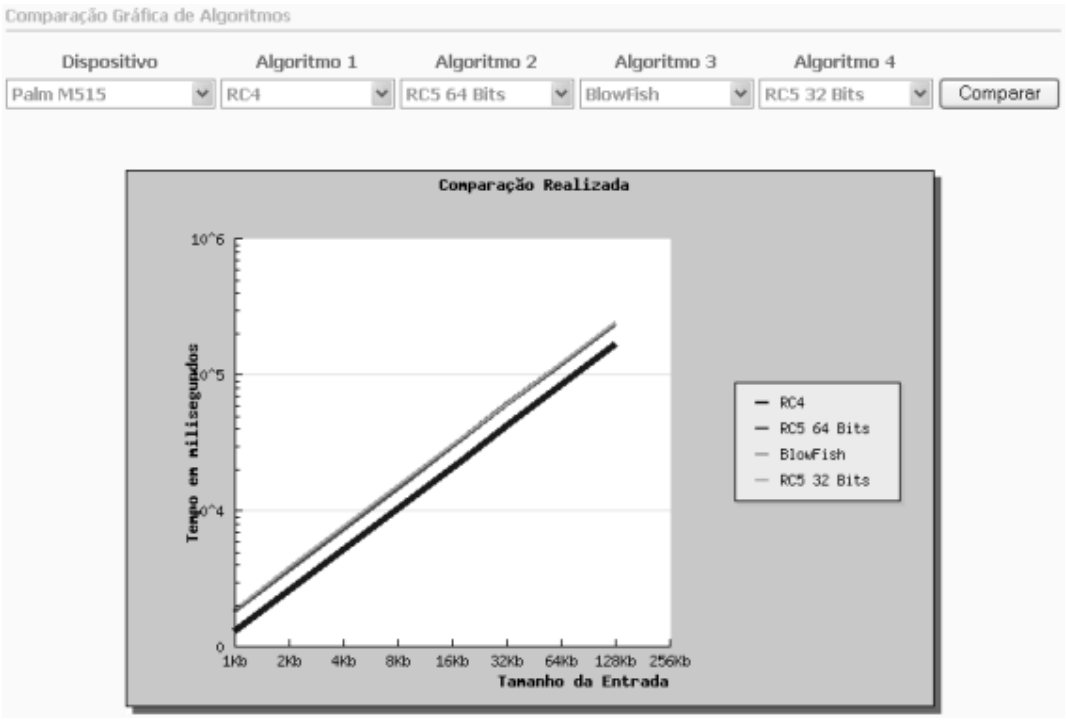

Figura 6.3. Gráfico comparativo dos quatro algoritmos de criptografia simétrica de melhor desempenho no DM Palm M515.

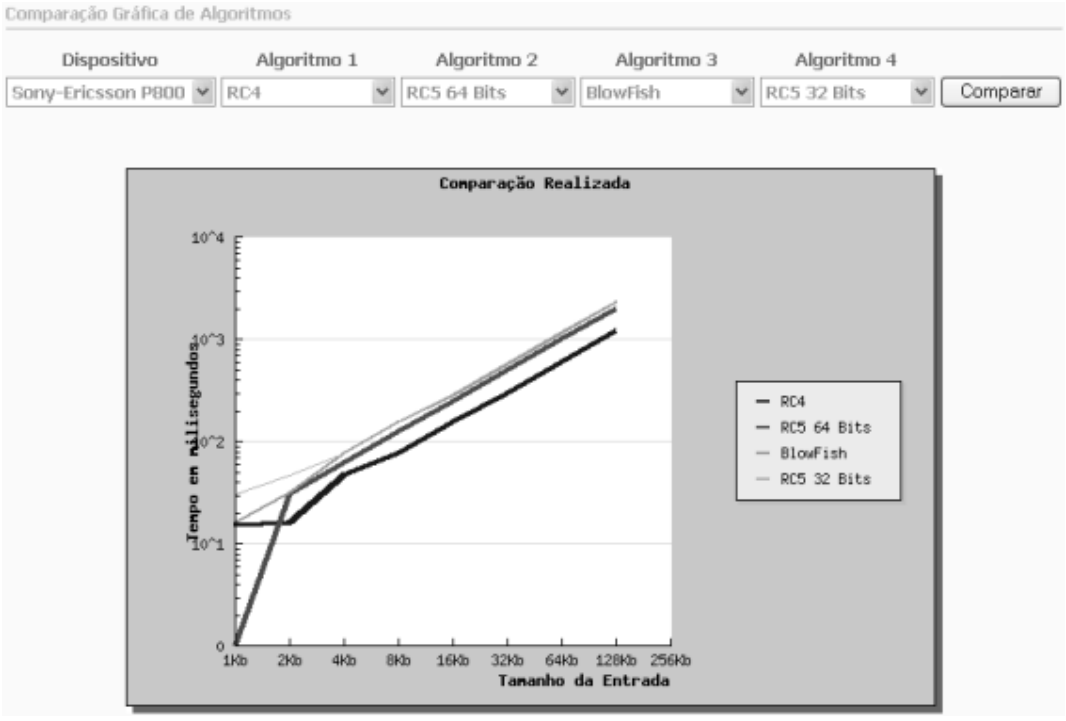

Figura 6.4. Gráfico comparativo dos quatro algoritmos de criptografia simétrica de melhor desempenho no DM Sony-Ericsson P800.

Tabela 6.3. Resumo das avaliações dos Algoritmos de Funções Hash.

Device	Minor throughput (Bytes/s)	Major throughput (Bytes/s)	Throughput increasing order
Palm M130	88,12	2546,07	MD2 < SHA256 < SHA512 <= SHA384 < SHA1 < RIPEMD160 < RIPEMD128 < Tiger < MD5 < MD4
Palm M515	87,9	2546,57	MD2 < SHA256 < SHA512 <= SHA384 < SHA1 < RIPEMD160 < RIPEMD128 < Tiger < MD5 < MD4
SonyEricsson P800	9309,1	2048000	MD2 < RIPEMD160 < SHA256 < SHA512 <= SHA384 < SHA1 < RIPEMD128 < MD5 < Tiger < MD4

As figuras a seguir (Figura 6.5, Figura 6.6 e Figura 6.7) ilustram os gráficos comparativos entre as quatro funções hash que apresentaram os melhores desempenhos nos DMs analisados.

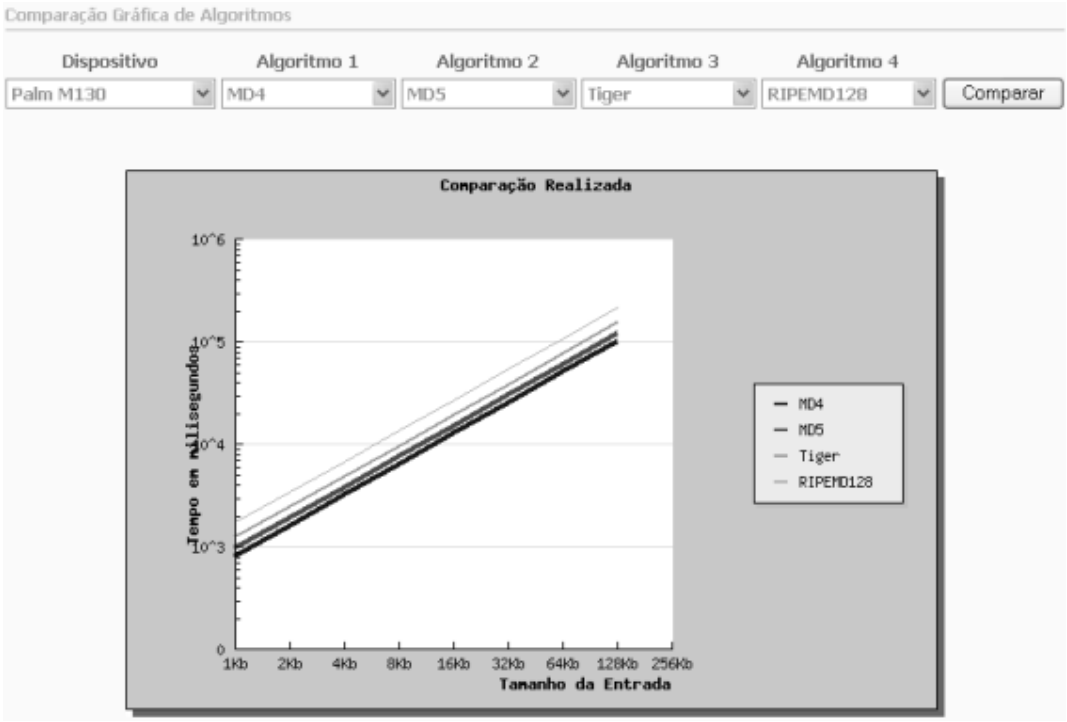

Figura 6.5. Gráfico comparativo das quatro funções hash de melhor desempenho no DM Palm M130.

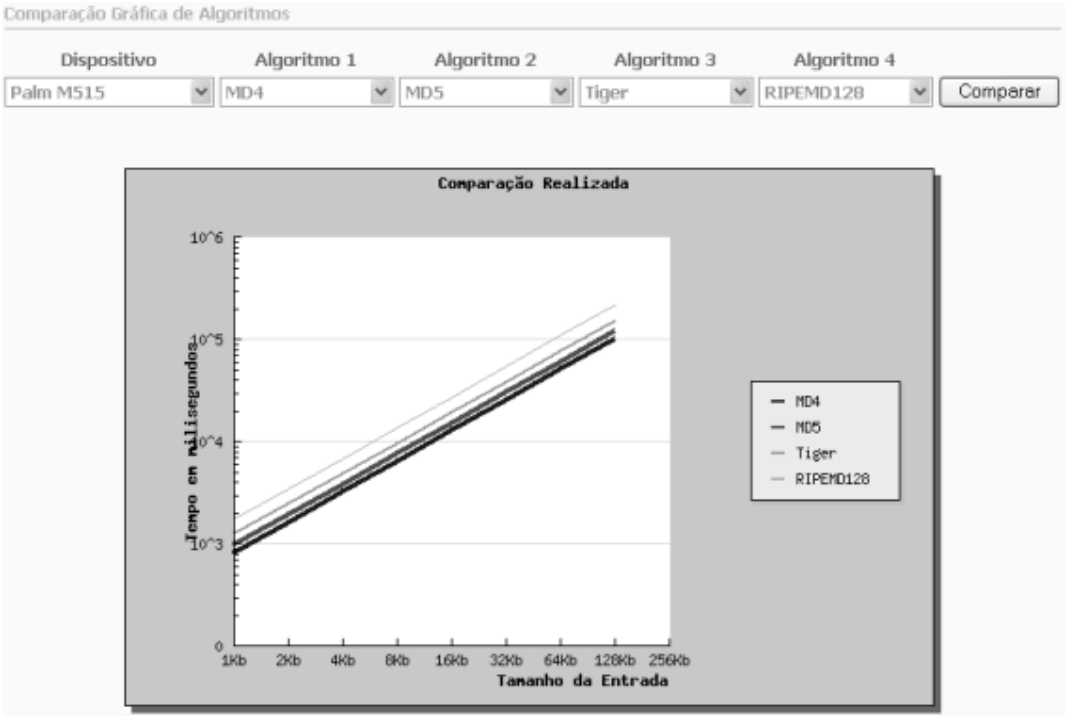

Figura 6.6. Gráfico comparativo das quatro funções hash de melhor desempenho no DM Palm M515.

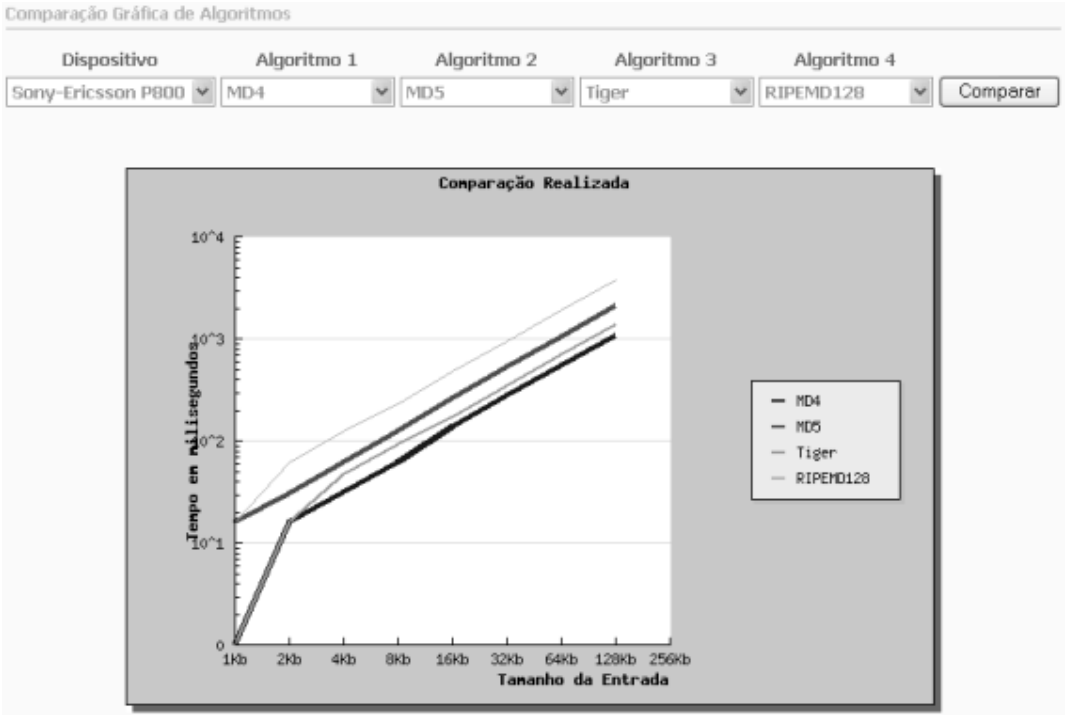

Figura 6.7. Gráfico comparativo das quatro funções hash de melhor desempenho no DM Sony-Ericsson P800.

Os resultados da análise nos DM demonstram que, apesar do dispositivo Palm M515 possuir o dobro da quantidade de memória do dispositivo Palm M130, o *throughput* dos algoritmos criptográficos nestes dispositivos estão bem próximos (veja a Tabela 6.2 e Tabela 6.3). Também é fácil observar que a ordem de crescimento do *throurghput* dos algoritmos nesses DM são iguais, diferindo-se da ordem de crescimento apresentada no dispositivo SonyEricsson P800. A próxima seção descreve o processo de escolha dos algoritmos que irão compor a instância do Framesec a ser incorporada à aplicação *MobileMulta*, levando-se por base os resultados das análises apresentados aqui.

6.2.2 Escolha dos algoritmos para os Hot Spots da instância do Framesec

A Tabela 6.4 foi montada a partir dos resultados apresentados na seção anterior. Nela estão listados os quatro algoritmos de criptografia simétrica e as quatro funções *hash* que apresentaram os melhores desempenhos nos DM analisados.

Tabela 6.4. Resumo das avaliações dos Algoritmos nos DMs.

Device	Throughput increasing order	
	Symmetric	Hash
Palm M130	RC5 32 Bits < BlowFish < RC5 64 Bits < RC4	RIPEMD128 < Tiger < MD5 < MD4
Palm M515	RC5 32 Bits < BlowFish < RC5 64 Bits < RC4	RIPEMD128 < Tiger < MD5 < MD4
SonyEricsson P800	BlowFish < AES Fast < RC5 64 Bits < RC4	RIPEMD128 < MD5 < Tiger < MD4

Levando-se por base os resultados obtidos através da análise, o desenvolvedor escolherá o algoritmo de criptografia simétrica RC4 e a função hash MD4 para compor os *hot spots* das instâncias do Framesec, para todos os DMs analisados. Entretando, conforme citado na Seção 5.4, somente o parâmetro de desempenho é insuficiente para garantir a construção de mecanismos seguros e eficientes. É necessário, portanto, estudar as vulnerabilidades existentes nestes algoritmos, de forma a permitir o balanceamento dos parâmetros desempenho e segurança dos algoritmos que irão compor as instâncias do Framesec. Por exemplo, o algoritmo RC4 possui vulnerabilidades de segurança, tais como as falhas presentes no

algoritmo de escalonamento de chaves KSA (*Key Scheduling Algorithm*) apresentadas em [18] e [19].

Este trabalho não desenvolveu o estudo da segurança oferecida pelos algoritmos criptográficos, ficando esta atividade para direcionamentos e trabalhos futuros. Portanto, para a escolha dos algoritmos que irão compor os *hot spots* do Framesec neste estudo de caso, foi considerado apenas os fatores de desempenho.

Os algoritmos de criptografia utilizados para compor os *hot spots* da instância do Framesec a ser incorporada na aplicação *MobileMulta* foi o RC4 e a função *hash* MD4. A próxima seção detalha o processo de instanciação do Framesec para a construção da aplicação *MobileMulta*.

6.3 Instanciando o Framesec para a construção da Aplicação MobileMulta

Conforme citado na Seção 6.1, a estratégia a ser utilizada na instância do Framesec é a *SecrecyIntegrityStrategy*, que provê a integridade e confidencialidade das informações transmitidas pela aplicação *MobileMulta*, até o SA localizado na central de trânsito (veja a Figura 6.1). Para permitir a decodificação das informações transmitidas pelo *MobileMulta*, a aplicação servidora localizada no SA deve também instanciar o Framesec, utilizando os mesmo algoritmos escolhidos pelo desenvolvedor para compor os hot spots da instância do Framesec a ser incorporada na aplicação *MobileMulta*.

Conforme citado na Seção 4.4, o Framesec não fornece suporte a mecanismos de troca de chave de forma segura. Portanto, o desenvolvedor deve implementar um mecanismo de troca de chaves para permitir o estabelecimento da chave de criptografia entre a aplicação *MobileMulta* hospedada no DM e a aplicação servidora localizada no SA.

No caso do *MobileMulta*, a distribuição de chaves ocorreu da seguinte forma: durante a instalação do MobileMulta a chave K gerada no servidor foi sincronizada com o DM através do cabo de dados. Uma cópia da chave K foi armazenada no banco de chaves localizada no SA, associada ao identificador do dispositivo ID. Portanto, a chave K foi estabelecida entre a aplicação *MobileMuta* e a aplicação servidora de forma segura.

Para aumentar ainda mais a segurança a chave K é trocada a cada semana, período em que o guarda de transito comparece à central para a manutenção do sistema.

Este período é determinado de acordo com as regras de negócio da aplicação, como forma de evitar ataques por meio da criptoanálise de várias mensagens catalogadas cifradas com a mesma chave (i.e., ataque *ciphertext only*).

A aplicação *MobileMulta* deseja utilizar a comunicação interprocesso transparente oferecida pelo Framesec. Portanto, a instância do Framesec inclui as classes *Peer*, *Forwarder*, *Receiver* e *Entry*, conforme citado na Seção 4.4.1. Para o desenvolvimento do *MobileMulta*, foi utilizada a plataforma J2ME CLDC/MIDP, que apresenta portabilidade com todos os DMs relacionados na Tabela 6.1. A Figura 6.8 ilustra o trecho de código, escrito em J2ME, do processo de instanciação do Framesec na aplicação *MobileMulta*.

```
// Código de instanciação do Framesec na aplicação MobileMulta
import Framesc.Peer;
import Framesec.Codifier;
import Framesec.Decodifier;
import Framesec.EntryHTTP;
import Framesec.ForwarderHTTP;
import Framesec.ReceiverHTTP;
import Framesec.parameter.SecrecyParams;
import Framesec.codifierstrategy.SecrecyIntegrityStrategy;
...
// Declaração de varíaveis
Peer mobileMulta;
EntryHTTP peers;
Codifier codifier;
Decodifier decodifier;
SecrecySymmetricParams params;
byte[] plaintext, ciphertext, key;
codifier = new Codifier(new SecrecyIntegrityStrategy("RC4", "MD4"));
decodifier = new Decodifier(new SecrecyIntegrityStrategy("RC4", "MD4"));
...
// Em algum momento, key recebe a chave estabelecida entre o DM e o SA
params = new SecrecySymmetricParams(key);

// inicialização do codifier e decodifier com os parâmetros
codifier.initialize(params);
decodifier.initialize(params);

// criando o banco de entradas
peers = new EntryHTTP();
peers.addEntry("SA", "http://200.17.37.12:8080", "post", codifier, decodifier);
mobileMulta = new Peer(new ForwardHTTP(peers), new ReceiverHTTP(peers));
```

Figura 6.8. Instanciando o Framesec na aplicação MobileMulta.

A aplicação *MobileMulta*, no momento em que desejar realizar o envio das multas cadastradas localmente no dispositivo, executa o trecho de código abaixo:

```
...
// recupera as multas cadastradas, e atribui ao plaintext
mobileMulta.send(plaintext, "SA");
```

Para receber as informações enviadas pelo SA, a aplicação *MobileMulta* executa o seguinte trecho de código:

```
...
// Recebendo a mensagem */
plaintext = mobileMulta.receive("SA");
```

Portanto, todo o processo de encriptação e decriptação das informações estão encapsuladas pelo Framesec, o que torna mais fácil e transparente a comunicação entre os DMs e o SA. O código implementado na aplicação servidora, que também instancia o Framesec, é similar ao apresentado na Figura 6.8, Dessa forma, não será apresentado neste estudo de caso.

6.4 Conclusão

Neste capítulo apresentamos como um desenvolver de uma aplicação, que necessite de segurança fim-a-fim, pode utilizar a ferramenta PEARL para compor os hot spots do Framesec. Com isso, esta dissertação não somente introduz um framework para o desenvolvimento de aplicações seguras para dispositivos móveis, como também propõe uma ferramenta para a avaliação de desempenho de algoritmos criptográficos. Com a aplicação MobileMulta demonstramos na prática a facilidade de uso integrado das duas contribuições principais deste trabalho.

7 Conclusão

Esta dissertação apresenta um *framework* para provisão de segurança fim-a-fim para o desenvolvimento de aplicações para dispositivos móveis. Além disso, uma ferramenta PEARL e o desenvolvimento da aplicação utilizando a ferramenta e o *framework* também são introduzidos neste trabalho.

7.1 Resultados Alcançados

Os protocolos de segurança existentes que operam nas camadas de rede e de transporte, quando não apresentam vulnerabilidades, são inviáveis de serem utilizados por dispositivos móveis que possuem limitações de recursos. Sendo assim, os mecanismos devem ser adicionados na camada de aplicação para a provisão de segurança fim-a-fim às aplicações para dispositivos móveis.

Estes mecanismos também devem ser construídos de acordo com os requisitos de segurança exigidos pela aplicação, que visam atingir os objetivos da criptografia, tais como confidencialidade, integridade, autenticidade e não-repudiação.

A construção desses mecanismos não é uma tarefa fácil, principalmente quando realizada por desenvolvedores de aplicações leigos em segurança da informação. A dificuldade é agravada, por que, geralmente, os requisitos de segurança não ocorrem de modo isolado, sendo necessário construir tais mecanismos de forma a atender combinações de requisitos de segurança distintos.

A construção de um *framework* que facilite o desenvolvimento aplicações com segurança fim-a-fim para dispositivos móveis é então necessária. O Framesec adiciona segurança na camada de aplicação, permitindo a transmissão segura de dados de forma independente do sistema de comunicação sem fio utilizado pelo dispositivo móvel.

Por outro lado, a avaliação de desempenho dos algoritmos criptográficos é um requisito básico para a construção de mecanismos de segurança eficientes em dispositivos de baixo poder computacional. A ferramenta PEARL apresentada neste trabalho permite avaliar o desempenho dos algoritmos criptográficos em uma vasta classe de DMs (e.g., Palms, celulares, Pocket PC), o que possibilita a identificação, de

forma satisfatória, dos algoritmos mais eficientes para um determinado dispositivo analisado.

Os resultados das avaliações de desempenho dos algoritmos no DM permitem ao desenvolvedor escolher, seguindo a escala de desempenho, os algoritmos que irão compor os *hot spots* da instância do Framesec. Portanto, a utilização da ferramenta permite a instanciação otimizada do Framesec ao DM que hospedará as aplicações construídas a partir do *framework*.

Portanto, os principais resultados alcançados neste trabalho são as propostas do Framesec para a provisão de segurança fim-a-fim para aplicações no ambiente de computação móvel, e da ferramenta PEARL, que permite avaliar o desempenho de APIs criptográficas que irão compor os pontos de flexibilização do Framesec, desenvolvidas na plataforma J2ME, em uma vasta classe de dispositivos móveis. O estudo de caso, que demonstra a integração entre as duas propostas, é também considerado um importante resultado alcançado através deste trabalho.

7.2 Trabalhos Futuros

Vale ressaltar que apenas o fator desempenho não garante a construção de mecanismos seguros. É necessário ainda um estudo da segurança oferecida pelos algoritmos escolhidos pelo desenvolvedor para compor a instância do Framesec. Isto por que a ferramente PEARL fornece apenas parâmetros de desempenho que auxiliam o desenvolvedor durante a escolha dos algoritmos, no entanto, a instância do Framesec otimizada em termo de desempenho pode não ser a mais segura.

Como resultado desse estudo, espera-se obter uma tabela de algoritmos ordenados por nível de segurança. Desta forma, o desenvolvedor poderá utilizar os resultados das avaliações de desempenho, bem como a tabela contendo os algoritmos criptográficos ordenados conforme os parâmetros de segurança definidos no estudo realizado, para realizar a escolha dos algoritmos que irão compor os hos spots do Framesec.

O Framesec proposto neste trabalho foi implementado na linguagem de desenvolvimento J2ME. Entretanto, o framework pode ser utilizado por qualquer plataforma de desenvolvimento Java, tais como J2EE, J2SE, PJava e Superwaba. Em trabalhos futuros, poder-se-ia implementá-lo em outras plataformas de

desenvolvimento, tais como C, assim como a ferramenta PEARL, com o intuito de abranger as plataforma de desenvolvimento de aplicações para DMs.

O Framesec proposto não fornece suporte a mecanismo de troca de chave, bem como a mecanismo de combinação de algoritmos que serão utilizados na instância do Framesec. Sendo assim, no momento da instanciação do Framesec o desenvolvedor deve informar quais algoritmos serão utilizados, bem como a chave que será utilizada. Portanto, outro direcionamento a trabalhos futuros seria adicionar suporte a estes mecanismos (troca de chave e combinação de algoritmos utilizados) ao Framesec.

Com relação a avaliação de desempenho, também é proposto a avaliação das instâncias do Framesec com o intuito de identificar o variação no desempenho adicionado em decorrência da utilização da estrutura do *framework*. Para tanto, são necessárias modificações na ferramenta PEARL de forma a permitir esta avaliação.

Além disso, propõe-se a disponibilização dos resultados das avaliações de desempenho à comunidade científica através da criação de um site da Web. Este site, além de centralizar e divulgar os resultados das avaliações já realizadas deve permitir novas inserções de resultados das avaliações de desempenho realizadas em novos dispositivos por outros desenvolvedores de aplicações espalhados no mundo.

For fim, durante a avaliação de desempenho dos algoritmos de criptografia, foram observadas pequenas variações estatísticas nos tempos de execução dos algoritmos, quando ocorria mudança do tipo de *plaintext* (e.g., texto ASCII, arquivo binário) avaliado. Portanto, é necessário um estudo aprofundado desses resultados, bem como da estrutura de funcionamento dos algoritmos visando identificar os fatores que possam ter ocasionado essas variações.

Referências Bibliográficas

[1] RACHERLA, G.; SAHA, D.; Security and Privacy Issues in Wireless and Mobile Computing. IEEE International Conference. p.17-20, dec, 2000.

[2] DIERKS, T.; ALLEN C. "The TLS Protocol version 1.0", IETF RFC 2246 Disponível em <http://www.faqs.org/rfcs/rfc2246.html>. Acesso em: 15 ago. 2003.

[3] GUPTA, V.; GUPTA, S. "Securing the Wireless Internet". IEEE Communications, Dec, 2001.

[4] DORNAN, ANDY. The Essential Guide to Wireless Communication Applications, Prentice Hall Inc., 2001. ISBN 0-13-031716-0.

[5] LOUREIRO, A.; MINELLI, A. Uma Ferramenta para Desenvolvimento de Aplicações para Dispositivos Móveis. Simpósio Brasileiro de Redes de Computadores, 20º, Búzios, v. 2, p. 571 - 586, 2002.

[6] ITANI, W.; KAYSSI, A.I.; J2ME End-to-End Security for M-Commerce.Wireless Communications and Networking - WCNC 2003. v. 3, p.16-20, mar, 2003.

[7] KENT, S.; ATKINSON, R. "Security Architecture for the Internet Protocol", IETF RFC 2401, Nov, 1998.

[8] DASWANI, N.; BONEH, D. "Experimenting with Electronic Commerce on the PalmPilot", Proc. Eurocrypt'99, LNCS 1648, Springer Verlag, p. 1-16, Fev, 1999.

[9] RAMSDELL, B. "S/MIME Version 3 Message Specification", IETF RFC 2633, June 1999.

[10] FERGUSON, N.; SCHNEIER, B. "A Cryptographic Evaluation of IPsec", Disponível em: <http://www.counterpane.com/ipsec.html>, Acesso em: fev, 2003.

[11] WAP Forum, "Wireless Transport Layer Security Specification". Disponível em <http://www.wapforum.org/what/technical.htm>. Acesso em: 10 ago. 2003.

[12] SORIANO M.; PONCE D., "A Security and Usability Proposal for Mobile Electronic Commerce," IEEE Communications, Ago, 2002.

[13] Wap Forum, "WAP 2.0 Technical White Paper". Disponível em <http://www.wapforum.org/what/WAPWhite_Paper1.pdf>. Acesso em: 15 ago. 2003.

[14] IEEE Std 802.11. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", mar. 1999.

[15] SCHWINDERSKI-GROSCHE, S.; KNOSPE, H. "Secure Mobile Commerce". Electronics & Communication Engineering Journal. Out, 2002.

[16] BROWN, B. "802.11 the security differences between b and i". Out, 2003.

[17] RSA Security. RSA Laboratories. Disponível em <http://www.rsasecurity.com>. Acesso em: 03 mar., 2003.

[18] FLUHRER, S.; MANTIN, I.; SHAMIR, A. "Weaknesses in the Key Scheduling Algorithm of RC4". SAC 2001.

[19] FLUHRER, S.; MANTIN, I.; SHAMIR, A. “Attacks on RC4 and WEP”. Cryptobyte 2002.

[20] 3G TS 21.101 3rd Generation mobile system Release 1999 Specifications (Release 1999).

[21] 3rd Generation Partnership Program. Document 1: f8 and f9 Specification. Technical Specification 35.201. Release 5. Version 5.0.0.

[22] 3rd Generation Partnership Program. Document 2: KASUMI Specification. Technical Specification 35.202. Release 5.Version 5.0.0.

[23] SILVA, G. M.; SOUZA, J. N.; Uma análise dos mecanismos de segurança de redes locais sem fio e uma proposta de melhoria. III Workshop de Segurança de Sistemas Computacionais, WSEG-SBRC 2003. Natal, maio, 2003.

[24] ARBAUGH, W.; WAN, Y.; SHANKAR, N. “Your 802.11 Wireless Network has No Clothes”, Disponível em <http://www.cs.umd.edu/%7Ewaa/wireless.pdf>. Acesso em: 20 mar. 2003.

[25] BORISOV, N.; GOLDBERG, I.; WAGNER, D. Intercepting Mobile Communications: The Insecurity of 802.11. 7th Annual International Conference on Mobile Computing and Networking, 2001.

[26] CAM-WINGET, N., HOUSLEY, R., WAGNER, D., WALKER, J. Wireless networking security: Security flaws in 802.11 data link protocols. Communications of the ACM, Volume 46 Issue 5, 2003.

[27] ROSHAN, P. “802.11 Wireless LAN Security White Paper”. Disponível em <http://www.cisco.com/en/US/products/hw/wireless/ps430/products_white_paper 0986a00800b469f.shtml>. Acesso em: 20 mar. 2003.

[28] “Port-based network access control”. IEEE Standard 802.1X. 2001.

[29] BLUNK, L; VOLLBRECHT, J.; "PPP Extensible Authentication Protocol (EAP)". RFC 2284.

[30] BUSCHMANN, F. Pattern-Oriented SoftwareArchitecture - A system of pattern. John Wiley, 1996.

[31] GAMMA, E. R. HELM; JOHNSON, R.; VLISSIDES J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1995.

[32] ANDERSON B., "Null Object", PloP '96.

[33] WATKINS, D. Overview and Comparison of GSM, GPRS and UMTS. Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, abr. 2000.

[34] BIRYUKOV, A., SHAMIR, A.; WAGNER, D. “Real Time Cryptanalysis of A5/1 on a PC”. abr., 2000. Disponível em <http://www.cryptome.org/a51-bsw.htm>. Acesso em: abr. 2003

[35] TARKKALA, L. “Attacks against A5”. Technical Report. Helsinki University of Technology, Department of Computer Science, 1999.

[36] BRICENO, M.; GOLDBERG, I.; WAGNER, D. “A pedagogical implementation of A5”, maio,1999. Disponível em <http://www.scard.org/gsm/a51.html>. Acesso em: abr., 2003.

[37] BRICENO, M.; GOLDBERG, I.; WAGNER, D. “A pedagogical implementation of A5/1 and A5/2”. Disponível em <http://cryptome.org/gsm-a512.htm>. Acesso em:abr., 2003.

[38] RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. The Unified Modeling Language Reference Manual. Addison-Wesley, 1998.

[39] PEREZ, P. A. S.; BOLDRINI, R.; DAHAB R. Aspectos de Segurança das Redes GSM, GPRS e UMTS. Minicurso, SBRC 2003. Natal, mai., 2003.

[40] PROMJIRAPRAWAT, K.; PIYATAMRONG, B. Secured Communication for GSM Networks. IEEE, 2003.

[41] ETSI TS 101 106. Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); GPRS ciphering algorithm requirements. European Telecomunications Standards Institute, mai., 2001.

[42] Bluetooth Special Interest Group, “The Bluetooth Specification, v.1.1” Disponível em <http://www.bluetooth.com/developer/specification/specification.asp>. Acesso em: 22, fev, 2002.

[43] JAKOBSSON, M.; WETZEL, S., “Security Weaknesses in Bluetooth”. Disponível em <http://www.rsasecurity.com/rsalabs/staff/bios/mjakobsson/bluetooth/bluetooth.pdf>. Acesso em: 20, mar, 2004.

[44] HAGER, C.T.; MIDKIFF, S.F.; An Analysis of Bluetooth Security Vulnerabilities.Wireless Communications and Networking, 2003. v. 3, p.16-20, mar., 2003.

[45] LOUGH, D. L. “A Taxonomy of Computer Attacks with Applications to Wireless Networks,” Ph.D. Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA. Disponível em: <http://scholar.lib.vt.edu/theses/available/etd-04252001-234145/>. Acesso em: abril, 2003.

[46] STALLINGS, W. Network Security Essentials: applications and standards. New Jersey: Prentice Hall, 1999.

[47] Menezes, A. J., Oorschot, P. C. V.; Vanstone, S. A.; Handbook of Applied Cryptography. Out, 1996.

[48] SUSILO, W. Securing Handheld devices. 10th IEEE International Conference, Wollongong, p. 349-354, 2002.

[49] RFC 2440 – “Open PGP Message Format ”, J. Callas, L. Donnerhacke, H. Finney, R. Thayer, Nov, 1998.

[50] Freundenthal, M.;Heiberg, S.; Willemson, J.; Personal Security Enviroment on Palm PDA. 16th Annual Conference, p. 366–372, Dec., 2000.

[51] Wong, D.S.; Chan, A.H.; Mutual authentication and key exchange for low power wireless communications. Military Communications Conference. V. 1, P. 39-43, 2001.

[52] WONG D. S.; CHAN A. H.; Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices, p. 1-18, dec, 2001.

[53] Shim, K;Cryptanalysis of Mutual Authentication and Key Exchange for Low Power Wireless Communications. IEEE , v. 7, mai., 2003.

[54] KINGPIN; MUDGE; Security Analysis of the Palm Operation System and its Weaknesses Againts Malicious Code Threats. 10th USENIX Security Symposium, Washingtion, p.135-151, Ago., 2001.

[55] KNIGHT W.; Palm Pilot open to a Denial of Service Attack. Disponível em <http://news.zdnet.co.uk/story/o,,s2074983,00.html>. Acesso em: 20 mar. 2003.

[56] WONG D. S.; FUENTES H. H.; CHAN A. H.; The Performance Measurement of Cryptographic Primitives on Palm Devices. Boston, p. 1-10, 2002.

[57] Argyroudis, P. G.;Verma, R.;Tewari, H.;O'Mayony, D. Performance Analysis of Cryptographic Protocols on Handheld Devices, 2003.

[58] HARRIS D.; The "Liberty" Crack: The first Palm OS Trojan Horse. Disponível em <http://www.sans.org/rr/pdas/sec_primer.php>. Acesso em: 20 mar. 2003.

[59] LEAVITT N.; Malicious code moves to mobile devices. Computer, p. 16-19, dec., 2000.

[60] @ STAKE. PalmOS Password Retrieval and Deconding. Disponível em <http://www.securiteam.com/securitynews/PalmOS_Password_Retrieval_and_Decoding.html>. Acesso em: 20 mar. 2003.

[61] HWANG T.; Scheme for Secure Digital Mobile Communications Based on Symmetric Key Cryptography. Tainan, p. 1-4, mar., 1992.

[62] WINKLE W. V.; Palm Business Applications. The Ultimate guide to mobile computing: 2002 mobile Technology. p. 82-89, 2002.

[63] Microsoft. Pocket PC Security. Disponível em <http://www.microsoft.com/mobile/enterprise/papers/security.aps>. Acesso em: 25 mar. 2003.

[64] McAfee.com. Virus Scan Wireless. Disponível em <http://www.mcafeeb2b.com/products/virusscan-wireless/default.asp>. Acesso em: 27 mar. 2003.

[65] PC-Cillin. Trend Micro PC-Cillin for Wireless 2.0. Disponível em <http://www.antivirus.com/download/documentation/netdesk/files/readme.txt>. Acesso em: 27 mar. 2003.

[66] MURTHY U.; BUKHRES O.; WINN W.; VANDERDEZ E.; Firewalls for Security in Wireless Networks. P. 1-9, 2000.

[67] ZWICKY E. D.; COOPER S.; CHAPMAN D. B; Construindo Firewalls para a Internet. Rio de Janeiro: Ed. Campus, 2001.

[68] Paragon Software. Powerful Crypto Protecion of your Windows CE devices. Disponível em <http://www.penreader.com/winCE/CryptoGrapher.html>. Acesso em: 10 abr. 2003.

[69] Roman, Steven; Desenvolvendo macros no Excel. Rio de janeiro: ED. Ciência Moderna, 2000.

[70] Wi-fi Alliance. Disponível em <http://www.wi-fi.org>. Acesso em: 15 abr. 2003.

[71] Java 2 Platform, Micro Edition (J2ME). Disponível em <http://java.sun.com/j2me>. Acesso em: 10 mar. 2003.

[72] PHP 4. Disponível em < http://www.php.net/>. Acesso em: 10 mar. 2003.

[73] Apache. Disponível em <http://www.apache.org/> Acesso em: 10 mar. 2003.

[74] MySQL. Disponível em < http://dev.mysql.com/> Acesso em: 10 mac. 2003.

[75] SonyEricsson. Disponível em <http://developer.sonyericsson.com/site/global/home/p_home.jsp>. Acesso em: 10 mar. 2003.

[76] Superwaba: The Java VM for PDAs. Disponível em < http://www.superwaba.com.br>. Acesso em: 10 mar. 2003.

[77] Microsoft. Embedded Visual Tools 3.0. Disponível em <http://www.microsoft.com/mobile/downloads/emvt30.asp>. Acesso em: 10 mar. 2003.

[78] VIANA, WINDSON; FILHO, BRINGEL; MAGALHÃES, KATY; GIOVANO, CARLOS; CASTRO, JAVAM DE; ANDRADE, ROSSANA. Mobis: A Solution For The Development Of Secure Applications For Mobile Device. In: ICT: International Conference on Telecommunications, 11th, Fortaleza-CE, Brazil. Proceddings to Appear, 2004.

[79] ANDRADE, R. M. C., Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems. 2001. 226 f. Thesis (Doctor of Philosophy in Computer Science)-School of Information Technology and Engineering (SITE), University of Ottawa-Carleton Institute of Computer Science, Ottawa, Ontario, Canada, 2001.

[80] ALBANO, W.; SALES, W.; ANDRADE, R.; CAVALCANTI, R.; SOUZA, J. N., SiGMA: Uma entidade para localização e autenticação de dispositivos móveis entre áreas de micromobilidade. XXII Simpósio Brasileiro de Redes de Computadores, maio, 2004.

[81] VIANA, WINDSON, FILHO, BRINGEL, CASTRO, R. M. C. PEARL: a PErformance evaluaAtor of cRyptographic aLgorithms for mobile devices. In: The First International Workshop on Mobility Aware Technologies and Applications (MATA 2004). Florianópolis-PR, Brasil, Outubro 2004 . Lecture Notes in Computer Science. MATA 2004 Conference: , v.3284,To Appear, 2004.BRAGA, A. M.; RUBINA C. M. F.; DAHAB, R.; Tropyc: A Pattern Language for Cryptographic Software. p. 1-27, jan, 1999.

[82] GUPTA, V.; GUPTA, S.; Experiments in Wireless Internet Security. Wireless Communications and Networking Conference – WCNC 2002. v. 2, p. 17-21, mar, 2002.

[83] SUSILO,W.; Securing Handheld devices. 10th IEEE International Conference, Wollongong, p. 349-354, 2002.

[84] FRIER A.; KARLTON P.; KOCHER P., “The SSL Protocol Version 3.0”. Disponível em <http://home.netscape.com/eng/ssl3>. Acesso em: 15 ago. 2003.

[85] PREE W.; **Design Patterns for Object-Oriented Software Development**. New York: Addison-Wesley Publishing Company, 1995.

[86] PREE, W. Hot spot Driven Development. In Fayad, M.E.; Johnson,R.E; Schmidt, D.C. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Wiley & Sons. p.379-393, 1999.

[87] FROEHLICH, G.; HOOVER, H; LIU, L; SORENSON, P. Reusing Hooks; In Fayad, M.E.,Schmidt, Douglas C; Building Application Frameworks: Object Oriented Foundations of Framework Design, 1999; pp 219-236.

[88] CARNEIRO C. M. P. S.; NETO M. G. M.; Frameworks de Aplicações Orientadas a Objetos – Uma Abordagem Iterativa e Incremental. II Simpósio Brasileiro de Qualidade de Sotware, Fortaleza, set., 2003.

[89] SILVA, R. P. Suporte ao Desenvolvimento e Uso de Frameworks e Componentes. 2000. 262 f. Tese (Doutorado em Ciência da Computação) – 96. Universidade Federal do Rio Grande do Sul, Instituto de Informática, Rio Grande do Sul.

[90] VILJAMAA, A.. Pattern-Based Framework Annotation and Adaptation – A systematic Approach, 2001. 118 f. Thesis (Licenciate Thesis in Computer Science) University of Helsinki, Finland.

[91] JOHNSON, R. E.; FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming, 1988. Disponível em:<ftp://st.cs.uiuc.edu/pub/papers/frameworks/designing-reusable-classes.ps>. Acesso em: 10 de maio 2003.

[92] LARMAN, C. Utilizando UML e Padrões Uma Introdução a Análise e ao Projeto Orientado a Objetos. Editora BookMan, 1999.

[93] GAMMA, E. R. H; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1995.

[94] TALIGENT WHITE PAPERS. Building Object Oriented Framework, inc, 1994. Disponível em <http://www.ipd.hk-r.se/michaelm/fwpages/fwbibl.html>. Acesso em: 20 de out. 2003.

[95] LAJOIE, R.; KELLER, R. K. Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts and Motifs in Concert; In: Proceedings of the 62nd Congress of the Association Canadienne Francaise pour l'Avancement des Sciences, Montreal, Canada, 1994 Disponível em: <ftp://st.cs.uiuc.edu/pub/patterns/papers/acfas.ps>. Acesso em: 15 de mai. 2003.

[96] JOHNSON, R. E. How to Design Framework. Notes for Proceedings of the Conference on Object-oriented programming system, languages, and applications -OOPSLA93, University of Ilinois, 1993. Disponível em: <ftp://st.cs.uiuc.edu/pub/papers/frameworks/OOPSLA93-frmwk-tut.ps>. Acesso em: 10 de maio 2001.

[97] Legion of The Bouncy Castle. Disponível em <http://www.bouncycastle.org>. Acesso em: 20 jun. 2003.

[98] NIST - National Institute of Standards and Technology. <http://www.nist.gov>. Acesso em: 20 mar. 2003.

[99] ETH Zürich - Eidgenössische Technische Hochschule Zürich. <http://www.ethz.ch/> Acesso em: 30 mar. 2003.

[100] DAEMEN, J.;GOVAERTS, R.;VANDEWALLE, J. Weak keys for IDEA, Advances in Cryptology - Crypto '93, Springer-Verlag (1994), 224-231.

[101] SCHNEIER, B.; KELSEY, J., WHITING, D., WAGNER, D., HALL, C., FERGUSON, N. The Twofish Encryption Algorithm. Wiley ISBN 0-471-35381-7, 1999.

[102] Official Blowfish website. Disponível em <http://www.schneier.com/blowfish.html> Acesso em: 20 mar. 2003.

[103] BRICKELL, E.F., DENNING, D.E., KENT, S.T., MAHLER, D.P., TUCHMAN, W."SKIPJACK Review", Interim Report. Disponível

em<http://www.cs.georgetown.edu/~denning/crypto/clipper/SKIPJACK.txt> Acesso em: 10 abr. 2003.

[104] ADAMS, C.M., “CAST Design Procedure Addendum". 1997.

[105] ELGAMAL, T. "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Transactions on Information Theory, v. IT-31, n. 4, pp469–472 *or* CRYPTO 84, pp10–18, Springer-Verlag, 1985.

[106] RFC 1319. The MD2 Message-Digest Algorithm. Disponível em <http:// http://www.faqs.org/rfcs/rfc1319.html> Acesso em: 29 abr. 2003.

[107] RFC 1186. The MD4 Message Digest Algorithm. Disponível em <http://www.faqs.org/rfcs/rfc1186.html> Acesso em: 29 abr. 2003.

[108] RFC 1320. The MD4 Message-Digest Algorithm. Disponível em < http://www.faqs.org/rfcs/rfc1320.html> Acesso em: 29 abr. 2003.

[109] The hash function RIPEMD-160. Disponível em <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html> Acesso em: 30 abr. 2003.

[110] A Fast New Cryptographic Hash Function. Disponível em <http://www.cs.technion.ac.il/~biham/Reports/Tiger/> Acesso em: 30 abr. 2003.

Apêndice A Segurança no dispositivo

Existem vários estudos relacionados à segurança nos DMs, como os descritos em [48][54][55][58][59][60]. Como os DMs estão vulneráveis a diversos tipos de ataques, é necessária a implementação de mecanismos de segurança a fim de protegê-los.

A segurança nesses dispositivos inclui prover proteção para o armazenamento de dados, para a comunicação sem fio, contra códigos maliciosos, para o sincronismo de dados, na conexão com a Internet, bem como fazer previsões para a perda ou roubo do dispositivo. Nas próximas seções detalharemos esses riscos à segurança a que os dispositivos móveis estão submetidos.

A.1 Armazenamento de dados

Os DMs podem armazenar localmente no dispositivos informações que devem ser mantidas em sigilo. Os sistemas operacionais suportados por esses DMs, tais como o Palm OS e Windows CE, não oferecem mecanismos para a proteção destes dados, sendo necessária a instalação de aplicativos de terceiros com este propósito.

Os DMs permitem ainda a expansão da capacidade de armazenamento através dos *slots* de expansão, por exemplo, o DM pode utilizar algum meio de armazenamento removível, tais como cartões *Compact Flash* (CF) ou *Secure Digital* (SD). As informações armazenadas nesses cartões também devem ser protegidas utilizando algum mecanismo de segurança que forneça encriptação dos dados.

Os mecanismos de proteção dos dados são fornecidos somente por aplicativos de terceiros, conforme mencionado anteriormente. Por exemplo, a ferramenta *CryptoGrapher* [68], que é uma solução para Pocket PC (Windows CE), provê esta proteção através da encriptação de todas as informação armazenadas, tanto no cartão quanto localmente.

A.2 A comunicação sem fio

Outro risco existente está relacionado à transmissão dos dados das aplicações desenvolvidas para esses dispositivos. Com a popularização dos DMs, as aplicações que antes eram confinadas aos prédios das corporações, passaram a ser hospedadas nos DMs

permitindo aos usuários obter acesso a qualquer momento e em qualquer lugar, eliminando as barreiras físicas existentes anteriormente. A migração das aplicações para o ambiente de computação móvel tem gerado preocupações quanto à segurança dos dados, já que o acesso indevido poderia ser realizado de fora das corporações por qualquer usuário que obtiver acesso físico ao DM [1].

Uma aplicação no ambiente de computação móvel normalmente envolverá DMs, a aplicação cliente, a aplicação servidora e um meio de comunicação para a transmissão dos dados entre a aplicação cliente e a aplicação servidora. A aplicação servidora será executada no SA.

A aplicação cliente deve ser desenvolvida em uma linguagem disponível para o sistema operacional utilizado pelo DMs (e.g., Palm OS, Windows CE e Symbian OS). Dentre as linguagens disponíveis, as mais utilizadas são: J2ME (Java 2 Micro Edition) [1], SuperWaba [76], C, Embedded Visual C++ (eVC) e Embedded Visual Basic (eVB) [77].

O SA está conectado à Internet, estando sujeito a ataques externos que podem causar interrupção de serviços, o que caracteriza um ataque à disponibilidade. A proteção do servidor pode ser alcançada através da implantação de um Firewall [65][66], bem como através da utilização de políticas de segurança de acesso aos serviços disponibilizados por ele.

Os dados transmitidos pelos DMs podem trafegar por meios de comunicação inseguros até chegar ao SA. Os meios de comunicação envolvidos na transmissão são diferentes de acordo com o tipo de conexão utilizado, conforme descritos no Capítulo 2. As informações transmitidas pelos dispositivos podem ser interceptadas, o que coloca em risco a confidencialidade dos dados. Estas mesmas informações interceptadas podem ainda ser modificada. Por exemplo, um atacante poderá modificá-las e posteriormente enviá-las ao SA. A aplicação servidora, ao receber as informações supostamente enviadas pelo usuário válido, tratará tais informações como verdadeiras. Esta situação caracteriza um ataque à integridade e dos dados.

Outro risco está ligado à fabricação de informações. Por exemplo, um atacante, passando-se por um usuário válido, poderá fabricar informações falsas e posteriormente enviá-las ao SA. A aplicação servidora, ao receber as informações, não será capaz de identificar a falsidade das informações, pois a identidade do usuário fornecida pelo atacente é uma identidade válida. Esse é um ataque a autenticidade e não-repudiação, os quais são citados no [46].

A.3 Códigos maliciosos

Os DMs estão se tornando alvo de códigos maliciosos, tais como Trojans e vírus. A instalação de aplicativos nesses DMs é realizada de forma trivial, não sendo necessário qualquer identificação do usuário que está realizando a instalação. Desta forma, a disseminação destes códigos em DMs é facilitada, como por exempo, em [58] é apresentado esse tipo de ataque em DMs que utilizam o sistema operacional Palm OS.

DMs cujo sistema operacional é o Windows CE, ainda não são alvos significante para esses códigos maliciosos [82]. Porém, com a evolução dos sistemas operacionais e dos aplicativos, será possível executar macros VBA (*Visual Basic Application* [69]) em aplicações do Pocket Office nesses DMs, o que os torna um novo alvo.

Além disso, os DMs podem ser utilizados para a disseminação de vírus de escritos para outras plataformas (i.e., sistemas operacionais). Por exemplo, um arquivo infectado com um vírus escrito para computadores *desktop* pode ser enviado ao DM. Ele não causará danos ao DM, pois a plataforma de execução do vírus é diferente da plataforma do DM. Em um momento futuro, este arquivo pode ser sincronizado com um computador *desktop* podendo causar a sua infecção, desde que o mesmo não possua proteção para o vírus (e.g., aplicativos antivírus). Programas antivírus estão sendo desenvolvidos para a plataforma de execução dos DM, tais como o McAfee VirusScan [64] e o PCCillin [65].

A.4 Sincronismo de dados

A transferência dos dados armazenados no DM pode ser realizada através da sincronização com um computador ou entre DMs. Durante este processo, é essencial verificar a autenticidade do usuário, evitando assim o acesso indevido a informações sigilosas armazenadas no DM.

A autenticação do usuário pode ser realiza através do par de informações *login* e senha, que devem ser informados antes do início da operação de sincronismo dos dados.

A.5 Conexão com a Internet

Para prover a capacidade de transmissão de dados aos DM, estes devem possuir conexão com a Internet de forma a permitir ao usuário obter acesso a informações a qualquer momento e em qualquer lugar.

Os DMs podem utilizar um dos sistemas de comunicação sem fio apresentados no Capítulo 2 para realizar a transmissão de dados, bem como: estabelecer conexões diretas via modem e telefone; através de uma rede local; ou via conexão TCP/IP (*Transmission Control Protocol/Internet Protocol*) com a ajuda de um computador, utilizando um cabo de dados.

Ao conectar um DM à Internet ele estará sujeito aos seguintes riscos:

- um ataque pode ser realizado diretamente ao DM ou ao computador no qual está conectado;
- o atacante poderá utilizar o DM como ferramenta para atacar outros computadores ou à rede a qual está conectado.

A.6 Perda ou roubo do DM

Devido ao seu tamanho pequeno, os DMs são facilmente perdidos ou roubados. O principal problema relacionado à ocorrência deste fato está ligado ao acesso indevido às informações armazenadas no DM, que pode conter informações sigilosas.

Por exemplo, o DM roubado pertencente a um executivo de uma empresa contendo informações das reuniões de lançamento de um novo produto, pode cair nas mãos dos seus concorrentes.

Os DMs, geralmente, fornecem proteção por senha de entrada como forma de evitar acessos indevidos. Por exemplo, os DMs com o sistema operacional Windows CE, conhecidos como Pocket PC, são protegidos por senha de entrada que é mantida em *hardware* [63]. Apesar da segurança fornecida através deste mecanismo, este apresenta vulnerabilidade. Geralmente, o tamanho da senha não é grande o suficiente para evitar ataques de força bruta, bem como o domínio normalmente inclui somente números, o que facilita ainda mais a sua descoberta. Por exemplo, em DMs com o sistema

operacional Palm OS, o mecanismo de senha é realizado em *software* e está vulnerável a este tipo de ataques, como demonstrado em [60].

Apêndice B Algoritmos Criptográficos

O presente apêndice se propõe a definir conceitos relacionados à criptografia, bem como apresentar os algoritmos criptográficos utilizados na implementação do Framesec na linguagem J2ME e na ferramenta PEARL. Os algoritmos criptográficos são divididos por grupo de acordo com as características específicas e finalidades a que se propõem. Os grupos são: criptografia simétrica, criptografia assimétrica e funções *hash*.

Os algoritmos apresentados neste apêndice podem ser utilizados de forma isolada ou em combinação com outros algoritmos, para a construção de mecanismos de segurança de acordo com os requisitos de segurança definidos pela aplicação que os utiliza. No caso em questão, estes serão utilizados para compor os pontos de flexibilização das instâncias do Framesec proposto neste trabalho, para a criação de aplicações no ambiente de computação móvel que possuem requisitos de segurança fim-a-fim.

B.1 Criptografia Simétrica

Os algoritmos de criptografia simétrica são caracterizados por utilizar uma chave única para realizar o processo de encriptação e decriptação [47], ou seja, a mesma chave utilizada para encriptar uma informação será utilizada para decriptá-la. Os algoritmos de criptografia simétrica são conhecidos também como algoritmos de criptografia de chave secreta ou de chave única, pois a segurança fornecida pelo algoritmo está concentrada no sigilo da chave. Esses algoritmos são utilizados com a finalidade de fornecer a confidencialidade das informações utilizando técnicas de encriptação.

Os algoritmos de criptografia simétrica são classificados em cifras de bloco (*block cipher*) e cifras de fluxo (*stream cipher*). Os algoritmos *block cipher* realizam a encriptação das informações em blocos de tamanho fixos, que são definidos pelo próprio algoritmo. Por sua vez, os algoritmos *stream cipher* realizam a encriptação sobre cada símbolo do *plaintext* de entrada sepadamente. Sendo assim, os *stream cipher* são considerados como cifras de bloco de tamanho um.

Toda a segurança dos algoritmos de criptografia simétrica está concentrada no sigilo da chave de criptografia utilizada, já que os algoritmos de criptografia, em sua grande maioria, são abertos para estudo. A velocidade do algoritmo está intrinsecamente

ligada ao tamanho da chave utilizada, o tamanho do arquivo de entrada e a quantidade de operações de transformações (substituição de um valor de entrada por outro) e transposições (modificação da posição de um valor) realizadas pelo algoritmo para encriptar as informações.

Nas próximas seções serão aprensentados os algoritmos de criptografia simétrica disponilizados no pacote de algoritmos de criptografia *Bouncy Castle* [97].

B.1.1 DES/Triple-DES

Baseado no algoritmo desenvolvido pela IBM chamado Lúcifer, o DES (*Data Encription Standard*) [47] foi primeiramente proposto em 1977. Em 1981, foi adotado como padrão pela ANSI com o nome de DEA (*Data Encryption Algorithm*), como tentativa de padronizar os procedimentos de encriptação no segmento privado. O DES é um algoritmo *block cipher* que utiliza blocos de tamanho de 64 bits e chaves de encriptação de 56 bits, que realiza a substituição monoalfabética do *plaintext* utilizando um alfabeto de 264 símbolos.

Como forma de aumentar a segurança do algoritmo DES, foi proposto por Tuchman o *Triple* DES (3DES ou DESede). Este algoritmo foi padronizado para uso em aplicações financeiras no padrão ANSI X9.17, em 1985. O 3DES utiliza chaves de 112 ou 168 bits, e realiza três execuções seguidas do algoritmo DES com chaves diferentes geradas a partir da chave de entrada.

B.1.2 RC2/RC4/RC5/RC6

Estas cifras foram projetadas por Ronald Rivest, da RSADSI (*RSA Data Security INC.*[17]). Elas permitem a utilização de chaves de tamanhos variados para proporcionar velocidade na criptografia de grandes quantidades de dados.

Estas cifras se tornam mais seguras escolhendo-se um tamanho de chave maior, podendo estar entre 1 e 2048 bits. O RC2 pode servir muito bem como um substituto para o DES, pois ambos são cifras do tipo *block cipher*. O RC4 é uma cifra do tipo *stream cipher*, desenvolvido em 1990, que é utilizada em protocolos de segurança em redes de computadores, tais como o WEP [14]. Este algoritmo foi mantido em sigilo até 1994, quando foi descoberto por técnicas de engenharia reversa. Esta versão não

oficial do algoritmo RC4 é distribuída utilizando o nome de "ARCFOUR", ou ARC4, devido ao nome RC4 ser registrado pela RSA.

O *block cipher* RC5, desenvolvido em 1994 e patentiado em 1998, permite a definição do tamanho da chave (0 a 2048 bits), o tamanho do bloco (32, 64 ou 128 bits) e o número de rodadas executadas pelo algoritmo de encriptação (0 to 255).

Por fim, o algoritmo RC6 é uma cifra do tipo *block cipher* baseado no RC5 que foi desenvolvida como requisito para a competição promovida pelo NIST (*National Institute of Standards and Technology* [98]) para a definição do algoritmo padrão AES (*Advanced Encryption Standard* [98]). Ele foi um dos finalistas desta competição, que teve como vencedor o algoritmo Rinjdael, que será descrito na próxima seção.

B.1.3 AES e o Rijndael

O algoritmo AES (*Advanced Encryption Standard* [98]), do tipo *block cipher*, possui tamanho de bloco de 128 bits e chaves de 128, 192 e 256 bits. Ele foi adotado pelo NIST [98] como padrão (US FIPS PUB 197) em novembro de 2001, depois de um longo processo de padronização que se estendeu por cinco anos.

O AES foi desenvolvido por dois criptográfos belgos, Joan Daemen e Vicente Rijmen. Ele não é baseado na cifra Feistel, como os demais finalistas da competição promovida pelo NIST para a escolha do algoritmo que seria utilizado neste padrão, conforme citado na Seção B.1.2. Ele tambpem é conhecido pelo nome original de submissão do algoritmo para a competição, chamado "Rijndael".

O AES não é precisamente o algoritmo Rijndael, porque este suporta tamanhos de bloco maiores que o definido na chamada de submissão realizada pelo NIST. Isto ocorreu devido à chamada de submissão do NIST para candidatos ao AES ter ocorrido após a criação do algoritmo Rijndael. Sendo assim, o Rijndael passou por adaptações, visto que a definição do AES utiliza tamanho de bloco fixo de 128 bits.

A implementação do AES é rápida, tanto em *software* quanto em *hardware*, além de requerer pouca memória. Por ser um novo padrão de *block cipher*, este algoritmo vem sendo largamente utilizado.

B.1.4 IDEA

O algoritmo IDEA (*International Data Encryption Algorithm*) é do tipo *cipher block*, desenvolvido na Suíça e publicado em 1990. O IDEA é a segunda versão do block cipher desenvolvido por Xuejia Lai e James L. Massey da ETH-Zürich [99] (*Swiss Federal Institute of Technology*).

O processo de encriptação do IDEA requer oito complexas rodadas. O decriptação é realizado da mesma forma que a encriptação, uma vez que as subchaves de decriptação foram calculadas a partir das subchaves de encriptação.

A estrutura da cifra do IDEA foi projetada para ser facilmente executada tanto em *software*, como em *hardware*. A segurança do algoritmo está na utilização de três tipos de operações aritméticas, incompatíveis entre si, que trabalham com palavras 16-bit. Entretanto, algumas das operações aritméticas usadas na IDEA não são rápidas quando implementadas em *software*. Como conseqüência, a velocidade do IDEA em *software* é similar à do DES.

O IDEA foi analisado pela RSA Security [17] quanto a força da cifra, onde foi submetido a técnicas de criptoanálise deferencial. Como resultado, o IDEA é imune a essa técnica. Até então, não existem ataques de criptoanálise linear no IDEA, bem como não há fraquezas algébricas conhecida. A única fraqueza conhecida foi descoberta por Daemen [100], ao utilizar uma uma classe de 2^{51} chaves fracas durante o encriptação resulta na deteção e na recuperação facilmente da chave. Entretanto, desde que há 2^{128} chaves possíveis, este resultado não tem nenhum impacto prático na segurança da cifra, já que a chave de encriptação é escolhida de forma aleatória. O IDEA é, portanto, considerada uma cifra muito segura. A estrutura bem como o embasamento teórico utilizados pela cifra foram abertamente discutido.

O IDEA é mais rápido e seguro que o DES. Por ser uma cifra recente (conseqüentemente não foi testada extensivamente) e ser patenteada, teve o seu uso comercial restringido. Este algoritmo é utilizado pelo PGP (*Pretty Good Privacy*) para a encriptação de arquivos e e-mail.

B.1.5 CAST5/CAST6

O algoritmo CAST5 [104] (também conhecido por CAST-128) é um algoritmo block cipher utilizado por padrão em algumas versões do GPG (*GNU Privacy Guard*) e PGP (*Pretty Good Privacy*). Ele também é aprovado pelo governo do Canadá para ser utilizado no estabelecimento de comunicações seguras. O CAST5 foi criado em 1996, por Carlisle Adams e Stafford Tavares. Por sua vez, o CAST6 (também chamado de CAST-256), derivado a partir do CAST5, foi um dos algoritmos candidados ao padrão AES promovido pela NIST.

O CAST5 executa 12 ou 16 rodadas, possui blocos de 64 bits e chaves de tamanho entre 40 e 128 bits. As 16 rodadas por completo são executadas somente quando o tamanho da chave é maior que 80 bits. Embora os procedimentos que compõem o algoritmo sejam patenteados, o CAST5 está disponível para utilização livremente em aplicações comerciais e não comerciais.

Por sua vez, o CAST6 utiliza blocos de 128 bits e tamanhos de chave 128, 160, 192, 224 ou 256 bits. O algoritmo executa 48 rodadas para encriptar e decriptar o *plaintext*.

B.1.6 Skipjack

O Skipjack [103] é um algoritmo do tipo *block cipher* desenvolvido pela *US National Security Agency* (NSA). Ele utiliza chaves de encriptação de 80 bits e blocos de 64 bits, realizando as operações criptográficas em 32 rodadas.

Eli Biham e Adi Shamir descobriram um ataque contra 16 das 32 rodadas executadas pelo algoritmo. Posteriormente, Alex Biryukov estendeu o ataque a 31 das 32 rodadas.

B.1.7 Serpent

O Serpent é um algoritmo do tipo *cipher block* que utiliza blocos e chaves de 128 bits, desenvolvido por Ross Anderson, Eli Biham e Lars Knudsen. Esta cifra executa uma rede de 32 rodadas de permutações e substituições, chamada (*SP-network*), operando sobre quatro blocos de palavras de 32 bits.

Este algoritmo foi um dos cinco finalistas da competição promovida pela NIST para a definição do padrão AES, conforme citado na Seção B.1.2, ficando em segundo lugar, depois do Rijndael.

O Algoritmo Serpent foi projetada de modo que todas as operações pudessem ser executadas em paralelo, utilizando 32 faixas de 1 bit. Portanto, foi estruturada de forma a maximizar o paralelismo.

B.1.8 Twofish

O Twofish [101] é um algoritmo do tipo *block cipher* que permite utilizar blocos e chaves de 128 ou 256 bits. Este algoritmo foi um dos cinco finalistas da competição promovida pela NIST para a padronização do AES, não sendo o algoritmo selecionado (veja na Seção B.1.2, o algoritmo selecionado).

Na maioria das plataformas de *software*, o Twofish é ligeiramente mais lento que o AES quando utiliza chaves de 128 bits. Entretanto, este algoritmo possui um desempenho melhor que o AES quando utiliza chaves de 256 bits.

B.1.9 Blowfish

O Blowfish [102] é uma cifra do tipo *block cipher*, desenvolvido em 1993 por Bruce Schneier. Este algoritmo utiliza blocos de 64 bits e chaves de tamanhos entre 32 e 448 bits. É uma das cifras mais rápidas de uso difundido, exceto quando realiza a mudança da chave de encriptação. A cada nova chave utilizada é requerido um pré-processamento equivalente a aproximadamente uma encriptação de um *plaintext* de 4Kb, considerado lento quando comparado a outras cifras do bloco. Isto impede a sua utilização em determinadas aplicações que efetuam constantes trocas de chaves (e.g., operações que utilizam chaves de seção com tempo de vida curto).

O Blowfish utiliza grande quantidade de memória, o que não é um problema para computadores *desktop*, mas impede a sua utilização em sistema *embedded*, tal como *smartcards*.

Assim como em outras cifras de bloco de 64 bits, não é aconselhável a utilização do Blowfish para encriptar *plaintext* grandes (maiores que 100 MB) com uma única chave, pois facilita a criptoanálise utilizando técnicas de texto cifrado conhecido.

O Blowfish, mesmo possuindo esta limitação, é uma cifra largamente difundida e utilizada.

B.2 Criptografia Assimétrica

A criptografia de chave pública, também conhecida como criptografia assimétrica [47], é baseada na utilização de pares de chaves para encriptar/decriptar as mensagens. O par de chaves são relacionadas através de um processo matemático que utiliza funções unidirecionais para a codificação da informação. Uma das chaves, chamada de chave pública (Ku), é utilizada para encriptar, enquanto a outra, chamada de chave secreta (Kr), é utilizada para decriptar.

Uma mensagem encriptada com uma chave pública só pode ser decriptada pela outra chave secreta com a qual está relacionada. A chave utilizada para encripar recebe o nome de chave pública porque ela é publicada por seu detentor aos nós com os quais ele se comunica. Por outro lado, a chave secreta (também chamada de chave privada) deve ser mantida em sigilo por seu detentor.

A criptografia assimétrica também pode ser utilizada para garantir a autenticidade do emissor. A este processo dá-se o nome de assinatura da mensagem, onde as informações enviadas são encriptadas utilizando a chave privada, e não mais a chave pública do nó de destino. Sendo assim, somente quem tiver a chave púbilca do emissor poderá recuperar a informação, e este terá a garantia de que a mensagem foi enviada pelo emissor, pois somente ele possui a chave privada. Portanto, a segurança da criptografia assimétrica está concentrada na segurança das chaves privadas, que devem ser mantidas em sigilo.

A criptografia assimétrica é mais lenta que a criptografia simétrica. Isto ocorre devido à complexidade das operações criptográficas utilizadas por esses algoritmos, que são baseadas na fatoração de números inteiros de múltipla precisão. Por outro lado, a segurança oferecida pela a criptografia assimétrica é superior à criptografia simétrica. Nas próximas seções, serão apresentados os dois algoritmos mais conhecidos de criptografia assimétrica e que estão presentes no pacote de criptografia *Bouncy Castle* [97].

B.2.1 RSA

O RSA [17] é um dos primeiros algoritmos de chave pública desenvolvidos no mundo, que foi descrito em 1977 por Ronald Rivest, Adi Shamir e Len Adleman. Clifford Cocks, um matemático britânico que trabalhava para GCHQ (*Government Communications Headquarters*), descreveu um sistema equivalente em um documento original interno, em 1973. Sua descoberta, entretanto, não foi revelada até 1997 devido a classificação do documento como sendo altamente confidencial. O RSA é utilizado tanto para a encriptação das informações (fornecendo a confidencialidade) quanto para a assinatura digital (fornecendo a autenticidade do emissor), conforme citado na seção anterior.

A definição do algoritmo RSA tomou por base o estudo realizado por Diffie e Hellman [47], porém, ele utiliza outro fundamento matemático para a criação das chaves públicas. Este fundamento se caracteriza pela facilidade de obter o resultado da multiplicação de dois números primos extensos. Entretanto, é estremamente difícil obter os fatores primos do número gerado pela multiplicação. Portanto, a segurança do RSA está na dificuldade de fatorar números inteiros muito grandes.

O tamanho da chave utilizada pelo RSA é variável, por exemplo, o RSA pode utilizar chaves de 256, 512 e 1024 bits. Entretanto, recomenda-se a utilização de chaves com tamanho igual ou superior a 1024 bits, por já ser possível quebrar a criptografia com chaves menores utilizando os recursos computacionais atuais, tais como ataques de força bruta distribuídos.

B.2.2 ELGamal

O algoritmo ElGamal [105] é baseado em operações de logaritmo discreto, sendo utilizado em softwares tais como o CGC (versão mais recente do PGP), bem como em outros sistemas. Ele pode ser utilizado tanto para a encriptação e decriptação das informações, como também para assinaturas digitais. O algoritmo de assinatura digital utilizado pela NSA (*National Security Agency*), chamado DAS (*Digital Signature Algorithm*) é baseado neste algoritmo e funciona de forma similar.

Acredita-se ser possível quebrar a cifra do ElGamal, entretando, é tão difícel quanto resolver o problema de logaritmo discreto. Se este problema pode ser resolvido

de forma eficiente, então o ElGamal pode ser quebrado. Entretanto, estudo recentes apontam a possibilidade de existir maneiras de quebrar a cifra do ElGamal sem ter que resolver este problema.

Quando uma mensagem é assinada utilizando o algoritmo ElGamal, o número aleatório utilizado para assinar a mensagem não pode se tornar público, assim como o mesmo valor aleatório não pode ser utilizado para assinar duas mensagens diferentes; fazer isso permitirá um oponente recuperar fàcilmente a chave secreta.

A próxima seção apresenta as funções *hash* utilizadas para a provisão da integridade e autenticidade da mensagem.

B.3 Funções Hash

A execução de uma função *hash* [47] assegura que o tamanho da saída, chamado de *digest* ou *message digest*, é constante, independente do tamanho da entrada fornecida ao algoritmo da função. A função garante também que, se a informação é alterada (até mesmo um bit), um valor de saída diferente será produzido, o que permite descobrir qualquer modificação na mensagem realizada por um oponente. As funções matemáticas utilizadas nos algoritmos das funções *hash* devem ainda garantir que não existem duas entradas distintas que fabriquem o mesmo *message digest* de saída.

As funções hash são classificadas em funções chaveadas e não chaveadas. As funções *hash* não chaveadas, também conhecidas como algoritmos de *Modification Detection Codes* (MDCs), não necessitam de chave de criptografia para calcular o *digest* da mensagem. Esse tipo de função é utilizado para garantir a integridade da mensagem. Por sua vez, as funções *hash* chaveadas, também chamadas de algoritmos de *Message Authentication Code* (MAC), necessitam de uma chave de encriptação para calcular o *digest* da mensagem. Dessa forma, a utilização deste algoritmo garante a autenticidade da mensagem, pois somente quem compartilha da chave de encriptação poderá calcular o *digest* da mensagem. Alguns algoritmos MAC são baseados na utilização de algoritmos de criptografia do tipo *block ciphers*, como será visto nas próximas seções.

B.3.1 MD2, MD4 e MD5

O MD5 (*Message-Digest Algorithm* 5) é um dos algoritmos MDC mais utilizados no mundo para garantir a integridade da mensagem. Foi desenvolvido por Ronald Rivest da *RSA Data Security* [17], e pode ser utilizado livremente em aplicações comerciais e não comerciais. O funcionamento deste algoritmo é baseado no algoritmo MD4 que, por sua vez, foi baseado no MD2. As funções *hash* não chaveadas MD2, MD4 e MD5 produzem *digest* de 128 bits a partir de qualquer tamanho de texto de entrada. Ronald Rivest publicou o MD2 na RFC1319 [106]. Com o intuito de criar um *digest* mais rapidamente que o criado através do algoritmo MD2, Ronald Rivest desenvolveu o MD4 que foi publicado nas RFC 1186 [107] e 1320 [108].

O MD5 é uma melhoria do seu predecessor, MD4, criado em resposta a trabalhos analíticos que indicaram que o MD4 é criptograficamente inseguro. O MD5 foi construído com o intuito de ser criptograficamente seguro. Entretanto, foram descobertas fraquezas que fazem o uso do MD5 questionável. Especificamente, mostrou-se que pares especiais de mensagens podem gerar o mesmo *digest*.

Os algoritmos MD2, MD4 e MD5 funcionam basicamente da seguinte forma: inicialmente o texto é dividido em blocos de tamanho fixo e depois é realizada, sucessivamente, uma série de operações matemáticas nos blocos de entrada.

B.3.2 SHA-1

O SHA-1 é uma função *hash* não chaveada desenvolvida pela NSA (*National Security Agency*) e publicada pela NIST [98]. O SHA-1 (*Security Hash Algorithm*) possui funcionamento similar ao das funções MD4 e MD5 desenvolvidas por Ronald Rivest, exceto que o SHA produz uma saída de 160 bits ao invés dos 128 bits de saída produzidos por esses algoritmos.

A especificação original do algoritmo foi publicada em 1993 (PUB FIPS 180) como o padrão de hash segura. Esta versão original é referenciada atualmente como SHA-0. Foi retirada de utilização pelo NSA logo após a publicação da versão substituta revisada SHA-1. A revisão foi realizada, de acordo com o NSA, para corrigir uma falha no algoritmo original que reduzia a segurança criptográfica da função *hash*. Entretanto, o NSA não forneceu detalhamento adicional a respeito.

Em 1998, na conferência Crypto, dois pesquisadores franceses, F. Chabaud e A. Joux apresentaram um ataque ao SHA-0. Este ataque não é aplicável ao SHA-1. Ainda não existem falhas de segurança no SHA-1 publicadas, sendo ainda considerado um algoritmo seguro.

O NIST publicou três variantes adicionais do SHA, cada um gerando *digests* mais longos. Estes são nomeados conforme o tamanho do *digest* gerado: SHA-256, SHA-384, e SHA-512. Foram publicados inicialmente em 2001 (PUB FIPS 180-2), no qual revisões e comentários foram aceitos. Esta publicação, que também incluia o SHA-1, foi liberada como um padrão oficial em 2002. Estas novas funções variantes do SHA não receberam ainda atenção por parte da comunidade criptográfica como o SHA-1, portanto estas funções ainda não são consideradas criptograficamente seguras.

B.3.3 RIPEMD

O RIPEMD-160 (*RACE Integrity Primitives Evaluation Message Digest*) [109] é uma função *hash* não chaveada que gera *digest* de tamanho de 160 bits, desenvolvida por Hans Dobbertin, Antoon Bosselaers e Bart Preneel, e publicado em 1996. O RIPEMD-160 é uma versão melhorada do RIPEMD que é baseado, por sua vez, nos mesmo princípios matemáticos utilizados no algoritmo MD4 e é similar em resistência e desempenho ao SHA-1.

Existem também outras versões deste algoritmo que geram *disgest* nos tamanhos de 128, 256 e de 320 bits, chamados RIPEMD-128, RIPEMD-256, e RIPEMD-320, respectivamente. A versão de 128 bits foi desenvolvida apenas para subistituir o RIPEMD original que também gera *digest* de 128 bits e já havia sido encontrados falhas de segurança neste algoritmo. As versões de 256 e 320 bits diminuem somente a possibilidade de colisão acidental, não possuindo, portanto, níveis mais elevados de segurança em comparação ao RIPEM-128 e o RIPEM-160, respectivamente.

B.3.4 Tiger

O *Tiger* [110] é uma rápida função *hash* não chaveada, projetada para ser utilizada em computadores modernos de 64 bits (tal como o DEC-Alfa). O *Tiger* não possui patente nem limitações de uso, podendo ser utilizada livremente em aplicações comerciais e não

comerciais. Este algoritmo foi projetado por Ross Anderson e Eli Biham em 1996, gerando *digest* de 192 bits.

Como foi desenvolvido para ser utilizado por computadores de 64 bits, quando utilizado em computadores de 32 bits o *Tiger* apresenta desempenho inferior, como pode ser visto através de exemplos de utilização apresentados em [110]. O *Tiger* ainda não possui ataques de segurança conhecidos e divulgados na literatura, no entanto, isto não garante que este algoritmo seja criptograficamente seguro.

B.3.5 HMAC

O HMAC (*Hash Message Authentication Code*) [47] é um algoritmo de autenticação de mensagem chaveado que combina as funções *hash* com funções dos algoritmos de criptografia simétrica. Assim como todo algoritmo MAC, este algoritmo pode ser utilizado para verificar simultaneamente a integridade e a autenticidade da mensagem.

O HMAC pode utilizar qualquer função *hash* para garantir a integridade do MAC calculado, como por exemplo, as funções SHA-1 e RIPEMD-160; a segurança do HMAC está diretamente relacionada à segurança da função *hash* utilizada pelo algoritmo para calcular o MAC, bem como o tamanho da chave de criptografia utilizada.

B.3.6 Funções hash chaveadas baseadas em Block Ciphers

As funções *hash* chaveadas mais comumente utilizadas são baseadas em algoritmos de criptografia do tipo *block cipher* [47]. Atualmente, existem funções *hash* baseadas nos algoritmos de criptografia *block cipher* utilizando os modos de cifragem CFB (*Cipher Feedback*) e CBC (*Cipher Block Chaining*).

Estas funções utilizam qualquer algoritmo de criptogria simétrica, do tipo *block cipher* (veja Seção B.1), tais como o DES e o AES. Portanto, a segurança dos algoritmos MAC está diretamente relacionada à segurança do algoritmo de criptografia simétrica utilizado para calcular o MAC, bem como ao tamanho de chave utilizada.