Especificação e Projeto de um Protocolo de Criptografia com Gerenciamento de Chaves Distribuído

Daniel B. Faria *Antonio A.F. Loureiro*
Patricia N.B. Soares *José Marcos S. Nogueira*
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702, 30123–970 Belo Horizonte, MG
Email: {dbfaria,loureiro,patty,jmarcos}@dcc.ufmg.br

Resumo

Neste trabalho, é especificado e implementado um protocolo para distribuição de chaves apropriado para um ambiente de redes de computadores que possui dentre outras características um número reduzido de interações entre os nós e serviço de autenticação de origem e destino. A especificação foi feita seguindo príncipios de projeto de protocolos de comunicação o que mostrou ser de extrema valia. Foram executados testes de desempenho com o objetivo de mostrar a viabilidade de uso desse protocolo em aplicações reais.

Abstract

In this paper, we specify and implement a protocol for distribution of keys which has among other characteristics a reduced number of interactions with nodes and an authentication service between origin and end nodes. The specification was written according to communication protocol design principles which showed to be extreme valuable. We also implemented and tested the protocol to analyze its suitability to real applications

1 Introdução

O crescimento das redes de computadores, notadamente a Internet, traz consigo vários problemas. Uma das grandes preocupações advindas da era da comunicação em escala mundial é a segurança da informação, já que uma parcela significativa dos dados que circulam nas redes contém informação de valor financeiro ou dados sigilosos. Existem várias propostas para solucionar o problema da segurança, que vão de arquiteturas gerais, como a do modelo OSI, até soluções envolvendo a própria aplicação, como o sistema Kerberos e o programa PGP.

O principal mecanismo para implementar segurança é a criptografia. Para viabilizar o seu uso em redes de computadores, é preciso que haja um mecanismo para distribuir as chaves de maneira segura e automática, de preferência através da

própria rede. Em virtude disso, uma das funções mais importantes do gerenciamento de segurança é a distribuição de chaves.

Existem diversos protocolos para distribuição de chaves. No entanto, nenhum deles é apropriado para todas as aplicações e capaz de resolver todos os problemas. Cada protocolo apresenta vantagens e desvantagens e, de forma geral, os mais seguros são os mais complexos. (Na seção 4 são discutidos dois protocolos importantes e suas limitações.) Para ser apropriado ao uso em redes de computadores, um protocolo de distribuição de chaves deve ter algumas características básicas, como número reduzido de interações entre os nós, serviço de autenticação de origem e destino, entre outras.

Este trabalho tem dois objetivos. O primeiro é especificar e implementar um protocolo para distribuição de chaves com as características descritas acima, proposto genericamente em [8], seguindo princípios de projeto de protocolos de comunicação como descritos em [4]. Ao se estudar a proposta [8] usada neste trabalho verificou-se que existiam vários pontos que não estavam claros ou precisavam ser melhor definidos. Ao invés de se tentar fazer uma especificação formal usando uma "Técnica de Descrição Formal" (FDT–Formal Description Technique) como Lotos ou Estelle preferimos fazer uma especificação mais detalhada seguindo os princípios de projeto de protocolos discutidos longamente por Holzmann [4]. De posse dessa especificação detalhada, que inclui os aspectos principais de projeto de um protocolo, poderíamos escrever essa especificação em uma FDT.

O nosso segundo objetivo é fazer uma análise de desempenho do protocolo proposto. Para isso escrevemos uma aplicação "'talk-like" e medimos o tempo de execução de cada etapa principal do protocolo de criptografia com gerenciamento de chaves distribuído. Com isso, podemos ter uma idéia melhor do overhead introduzido pelo protocolo.

Este trabalho está organizado da seguinte forma. A seção 2 apresenta a especificação e projeto do protocolo proposto. A seção 3 discute a implementação efetuada e apresenta os testes executados. A seção 4 discute os trabalhos relacionados. Finalmente, a seção 5 apresenta as conclusões e discute trabalhos futuros.

2 O Protocolo KMP (Key Management Protocol)

Esta seção apresenta o protocolo usado neste trabalho. A seção 2.1 descreve o protocolo KMP. A seção 2.2 resume os princípios de projeto usados na especificação de acordo com [4]. Finalmente, a seção 2.3 apresenta a especificação e projeto do protocolo KMP.

2.1 Descrição do Protocolo

A proposta apresentada em [8] compreende um protocolo seguro de distribuição de chaves de sessão. O objetivo da utilização deste protocolo é que somente as entidades envolvidas na aplicação conheçam a chave criptográfica usada, o que torna a comunicação segura. Todos os dados de controle trocados devem também estar codificados.

Para que o protocolo proposto seja considerado satisfatório, ele deve prover segurança para as aplicações e ao mesmo tempo a sobrecarga (overhead) causada para o sistema deve ser minimizada, de forma que este possa ser amplamente utilizado.

Para aliar segurança e eficiência, o protocolo de gerenciamento de chaves (KMP) utiliza um algoritmo de chave pública para fazer a distribuição das chaves simétricas para comunicação e também para fazer a autenticação da origem e destino. Após a distribuição da chave de sessão, esta é usada para prover a confidencialidade dos dados através de um algoritmo simétrico de bloco que opera em modo Cypher Block Chainning (CBC). Este modo utiliza um vetor de inicialização IV que será enviado utilizando-se o mesmo algoritmo simétrico de bloco, agora operando em modo Eletronic Code Book (ECB). A geração das chaves e a autenticação dos clientes é realizada no Servidor de Autenticação e de Chaves (AKS), ou simplesmente servidor de chaves.

Para ilustrar o funcionamento do protocolo, vamos supor que dois nós da rede, A e B, desejam se comunicar seguramente. Existe um terceiro nó que centraliza e gerencia a distribuição das chaves de sessão para A e B. Este terceiro nó funciona como um Servidor de Autenticação e de Chaves (AKS) gerando e autenticando chaves de sessão criadas de forma aleatória. Através destes três nós podemos simular um caso que, embora simples, exemplifica exatamente o funcionamento do protocolo para um número qualquer de nós. O servidor mantém relacionados todos os clientes que podem se comunicar de forma segura. O conjunto formado pelo servidor e os clientes nele relacionados constitui a rede segura.

O protocolo poder ser dividido em duas fases: distribuição das chaves públicas e distribuição das chaves simétricas secretas (chaves de sessão). Na primeira fase são distribuídas as chaves públicas que serão usadas para distribuição das chaves de sessão. Esta fase equivale a registrar o nó no Servidor de Autenticação e Chaves e é executada quando a rede é inicializada ou quando um novo nó é adicionado à rede segura. Uma vez que um dos nós tem sua chave pública comprometida, esta fase também é executada. Na segunda fase há a distribuição das chaves de sessão e esta é executada sempre que for iniciada uma comunicação segura entre clientes. Para que os dados, entre os quais está a chave de sessão, sejam recebidos apenas pelos nós seguros, eles são cifrados com as chaves públicas correspondentes e assim somente os verdadeiros destinatários podem reconstruir os dados com suas chaves secretas.

2.1.1 Distribuição das Chaves Públicas

Esta fase é responsável pela geração de um cadastro eletrônico contendo todos os clientes que estejam na rede segura e suas respectivas chaves públicas. Este cadastro fica então armazenado no Servidor de Chaves que tem sua chave pública conhecida por todos os usuários. Depois da distribuição destas chaves públicas, deve haver uma autenticação delas para que o protocolo possa prosseguir seguramente, isto é, todos os nós juntamente com o AKS devem certificar que possuem as mesmas chaves. Isto deve ser feito manualmente e consiste na publicação, por parte do Servidor de Chaves, de valores de uma função hash conhecida por todos os nós aplicada a cada uma das chaves públicas. Assim, estes valores são publicados e verificados. Nesta fase preliminar, esta é a única tarefa que não é feita automaticamente. O funcionamento desta fase está exemplificada na figura 1.

Em primeiro lugar, o administrador da segurança do Servidor de Chaves e de

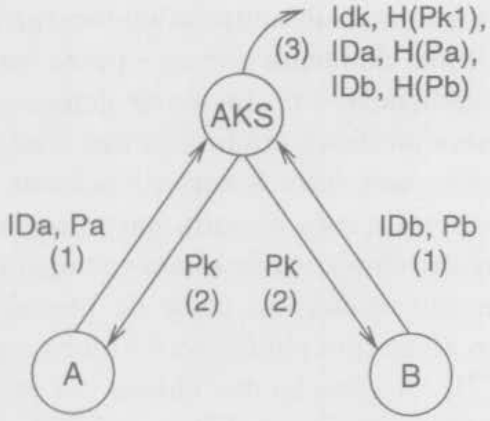

Figura 1: Distribuição das Chaves Públicas

Autenticação (AKS) coloca no AKS uma relação de todos os nós que fazem parte da rede segura. Em seguida, o AKS gera Sk e Pk e armazena Sk de forma segura. Cada nó da rede segura, aqui composta por A e B, age da mesma forma e cria seu par de chaves respectivo, armazenando sempre sua chave secreta de forma segura. Uma vez que tenham gerado seu par de chaves, A e B enviam suas chaves públicas para o servidor juntamente com seus identificadores. No recebimento, o AKS confere os identificadores, verifica se os nós realmente pertencem à rede segura, armazena suas chaves públicas e envia sua própria chave pública para A e B. O AKS e os nós A e B calculam os valores de uma função hash unidirecional pré-definida e conhecida por A, B e AKS para cada uma das chaves conhecidas. O administrador de segurança publica então os valores da função hash para cada uma das chaves públicas presentes no cadastro do AKS para verificação manual pelos administradores dos nós A e B. Nesse momento, A e B podem então verificar se a chave Pk que possuem é verdadeira e também se suas próprias chaves foram corretamente recebidas e publicadas pelo AKS. Este é o único procedimento manual e deve ter um prazo máximo pré-determinado, depois do qual o AKS libera a operação dos nós. O uso da função hash é importante uma vez que seu valor de saída é bem menor que a chave pública, o que facilita a verificação. Uma vez que estes valores tenham sido verificados, um nó na rede segura está pronto para iniciar a segunda fase.

2.1.2 Distribuição das Chaves de Sessão e Autenticação

O protocolo apresentado aqui é baseado no protocolo de distribuição de chaves de sessão e autenticação com hierarquia de dois níveis proposto em [8]. Na descrição feita a seguir, o nó A é o nó que vai iniciar a comunicação segura com o nó B. Para geração e distribuição da chave de sessão que será usada são necessários quatro passos (Figura 2).

Inicialmente, A envia seu identificador e o identificador de B para o AKS. O AKS gera aleatoriamente uma chave de sessão inicial Ksi e produz uma marca de tempo T que indica quando a chave foi gerada. Cifra então Ksi e T com as chaves públicas de A e B garantindo assim que somente A e B vão obter Ksi. Adicionando a estes dados as chaves públicas e os identificadores de A e B, e assinando tudo com sua chave secreta, o AKS cria o Ticket de Autenticação, ou simplesmente Ticket.

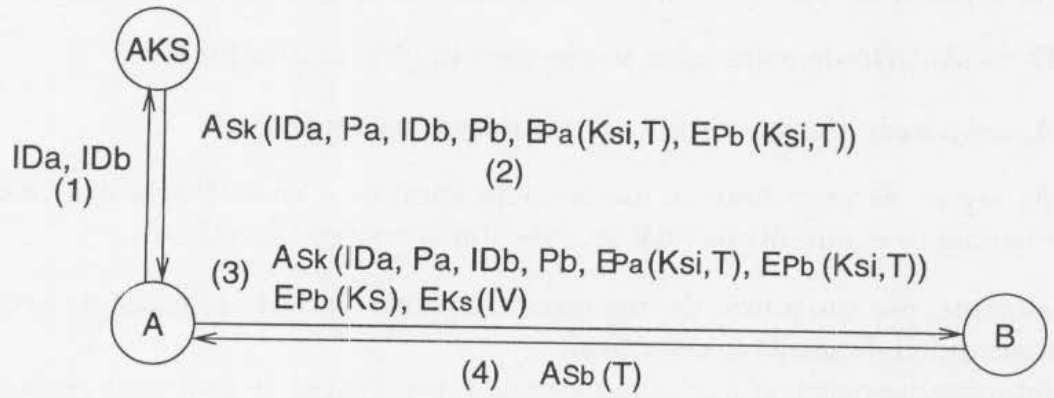

Figura 2: Distribuição da Chave de Sessão com Autenticação Completa

O AKS então envia o Ticket para A. A recebe o Ticket e o decifra, obtendo os diversos parâmetros e verifica se os identificadores IDa e IDb são os mesmos que foram enviados para o AKS. A verifica também T para saber que a chave de sessão não é repetida, verificando se T é menor que (T + dt), onde dt é o período de validade da chave determinado para a rede. Cifrando Ksi com sua chave Sa, A gera a chave de sessão efetiva.

Para que a chave de sessão criada seja utilizada pelo algoritmo simétrico em modo Cypher Block Chainning, A cria aleatoriamente o vetor de inicialização IV e o cifra com a chave de sessão Ks com o algoritmo simétrico em modo Eletronic Code Book. Para ser enviada, a chave de sessão Ks é crifrada com a chave pública de B (Pb). A envia então, para B, Ks e IV cifrados juntamente com Ticket recebido do AKS. Quando A cifra Ksi com sua chave secreta, produzindo Ks, torna a chave de sessão desconhecida também para o AKS, um dos pontos que difere este protocolo de outros trabalhos relacionados.

B decifra os parâmetros recebidos e retira Ks. Para verificar se Ks foi realmente gerada por A, B compara Ksi (retirada do Ticket) com o valor obtido com a decifragem de Ks com a chave pública de A. Para que A tenha certeza que B recebeu corretamente os dados enviados, B cifra a marca de tempo T recebida com sua chave secreta e envia para A. A calcula T através da decifragem com a chave pública de B e compara com o valor enviado. Se os valores forem iguais, A tem certeza que o último foi enviado por B e a autenticação fica completa. A e B possuem agora uma chave de sessão e o vetor de inicialização que serão usados na comunicação segura.

2.2 Princípios de Projeto

As descrições de protocolo encontradas normalmente na prática são informais o que faz com o projetista ou implementador do sistema tenha que fazer decisões muitas vezes sem nenhuma base ou conhecimento adequado. Um primeiro passo para evitar esse problema é estruturar e formalizar as descrições do protocolo de tal forma a tornar todas suposições explícitas.

Holzmann [4] identifica cinco partes distintas na especificação de qualquer protocolo de comunicação, a saber:

1. O *serviço* a ser provido pelo protocolo

2. As *suposições* sobre o *ambiente* no qual o protocolo será executado
3. O *vocabulário* de mensagens usado para implementar o protocolo
4. A *codificação* (formato) de cada mensagem no vocabulário
5. As *regras de procedimento* usadas para garantir a consistência das trocas de mensagens e, em última análise, executar o serviço especificado

Normalmente, por um abuso de linguagem, a quinta parte é chamada de protocolo e é a mais difícil de projetar e verificar.

É interessante observar que cada parte da especificação do protocolo pode definir uma hierarquia. Por exemplo, o vocabulário do protocolo pode ser formado por uma hierarquia de classes de mensagens.

De uma forma geral, o projeto do protocolo deve ser estruturado buscando sempre:

- *Simplicidade*—protocolo deve ser construído a partir de um pequeno número de funções bem projetadas e bem entendidas.
- *Modularidade*—um protocolo complexo pode ser construído a partir de módulos mais simples que interagem de forma bem definida e simples.
- *Bem-formado*—um protocolo, como qualquer outro sistema, não deve conter funções que nunca serão executadas ou que não foram definidas; deve possuir limites conhecidos como tamanho de fila de mensagens; deve ser auto-estabilizante [5]; pode ser adaptado.
- *Robusto*—idealmente, o protocolo deve fazer suposições mínimas sobre o ambiente onde será executado. Na prática, isso é difícil de obter pois o ambiente influencia diretamente a forma como o protocolo deve trabalhar.
- *Consistência*—protocolos, como outros algoritmos distribuídos, devem possuir certas propriedades como não possuirem deadlocks ou livelocks, terminações erradas.

Para o leitor interessado, Holzmann [4] discute muito bem a questão de projeto de protocolos. A seguir é apresentado a especificação do protocolo usando as cinco partes descritas acima.

2.3 Especificação do Protocolo

O modelo de referência que será usado neste texto é uma combinação dos modelos OSI e TCP/IP, estando representado pelas seguintes camadas: física, enlace, rede, transporte e aplicação. O protocolo de gerenciamento de chaves (KMP) está localizado na camada de aplicação, de forma que utiliza os serviços do protocolo TCP, presente na camada de transporte. É importante definir o conceito de um cliente seguro no protocolo que será proposto. Na exemplificação do protocolo dado na seção 2.1, o conceito de cliente seguro está diretamente relacionado a um nó da rede, ou seja, uma máquina. Desta forma, uma máquina teria, para todas as aplicações que utilizam o protocolo de gerenciamento de chaves, um mesmo par de chaves pública e privada, o que não é uma medida segura. Na arquitetura TCP/IP, uma aplicação está ligada a um determinado porto, de forma que o conjunto (endereço IP + porto) define assim um candidato a cliente seguro. Com esta medida, o identificador de um cliente seria justamente esta dupla (IP,porto), de forma que mesmo

que várias aplicações utilizem o protocolo de gerenciamento de chaves, cada uma terá seu par de chaves independente. A figura 3 mostra a localização do protocolo KMP no modelo de referência usado.

Figura 3: Modelo de Referência Utilizado

2.3.1 Primitivas de Serviço

O protocolo KMP oferece um conjunto de primitivas de serviço para envio e recepção de mensagens de gerenciamento. Pode-se dividir o protocolo KMP em dois módulos distintos, KMPc e KMPs, que são executados num nó cliente e no Servidor de Chaves e Autenticação, respectivamente. Abaixo encontram-se as descrições das primitivas do protocolo.

- **Identification.req**
  Direção: Usuário → KMPc
  Descrição: Esta primitiva é invocada quando o cliente deseja cadastrar seu identificador e sua chave pública no AKS.
  Parâmetros:
  - Identificador do cliente: Tupla formada pelo endereço IPv6 do nó onde o cliente está sendo executado e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
  - Chave pública do cliente: vetor de 92 bytes que representa esta chave pública.

- **Identification.ind**
  Direção: KMPs → Servidor
  Descrição: Esta primitiva informa o Servidor de Chaves que um pedido de cadastramento foi recebido.
  Parâmetros: retorna os parâmetros enviados em Identification.req.

- **Identification.resp**
  Direção: Servidor → KMPs

Descrição: Esta primitiva é invocada pelo AKS para enviar uma resposta a um pedido de cadastramento feito por um cliente.

Parâmetros:
- Identificador de Erro: um número inteiro que indica se houve algum erro de cadastramento identificado pelo Servidor de Chaves. Este parâmetro consiste em um número inteiro entre 0 e 255;
- Chave pública do Servidor de Chaves: vetor de 92 bytes que representa esta chave pública.

- **Identification.conf**

Direção: KMPc → Usuário

Descrição: Esta primitiva informa o cliente que uma resposta ao pedido de cadastramento foi recebida.

Parâmetros: retorna os parâmetros enviados em Identification.resp.

- **OpenSession.req**

Direção: Usuário → KMPc

Descrição: Esta primitiva é invocada pelo cliente quando este deseja iniciar uma sessão segura com outro cliente.

Parâmetros:
- Identificador do cliente de origem: tupla formada pelo endereço IPv6 do nó onde está o cliente que solicita a sessão e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Identificador do cliente de destino: tupla formada pelo endereço IPv6 do nó onde está o cliente que irá participar da sessão juntamente com o cliente origem e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1.

- **OpenSession.ind**

Direção: KMPs → Servidor

Descrição: Esta primitiva informa o Servidor de Chaves que um pedido de abertura de sessão foi recebido.

Parâmetros: retorna os parâmetros enviados em OpenSession.req.

- **OpenSession.resp**

Direção: Servidor → KMPs

Descrição: Esta primitiva é invocada pelo AKS para enviar uma resposta a um pedido de abertura de sessão feito por um cliente. O conjunto dos parâmetros abaixo forma o Ticket de Autenticação.

Parâmetros:
- Identificador de Erro: um número inteiro que indica se houve algum erro no pedido de abertura de sessão identificado pelo Servidor de Chaves. Este parâmetro consiste em um número inteiro entre 0 e 255;

- Identificador do cliente de origem: tupla formada pelo endereço IPv6 do nó onde está o cliente que solicita a sessão e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128}-1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Chave pública do cliente origem: vetor de 92 bytes que representa esta chave pública.
- Identificador do cliente de destino: tupla formada pelo endereço IPv6 do nó onde está o cliente que irá participar da sessão juntamente com o cliente origem e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128}-1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Chave pública do cliente destino: vetor de 92 bytes que representa esta chave pública;
- Chave de sessão inicial: número inteiro entre 0 e $2^{64}-1$ que representa a chave de sessão inicial que será utilizada na sessão solicitada;
- Marca de Tempo: número inteiro entre 0 e $2^{32}-1$ que representa o instante de tempo no qual a chave de sessão inicial foi gerada.

- **OpenSession.conf**
  Direção: KMPc → Usuário

  Descrição: Esta primitiva informa o cliente que uma resposta ao pedido de abertura de sessão foi recebida.

  Parâmetros: retorna os parâmetros enviados em OpenSession.resp.

- **WakeUp.req**
  Direção: KMPc → Usuário

  Descrição: Esta primitiva informa o cliente destino que ele é chamado em uma sessão e retorna um valor que indica uma resposta deste com respeito à abertura da sessão.

  Parâmetro:
  - Identificador do cliente de origem: tupla formada pelo endereço IPv6 do nó onde está o cliente que solicita a sessão e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128}-1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1.

- **WakeUp.ind**
  Direção: Usuário → KMPc

  Descrição: Esta primitiva informa o KMPc sobre o estado do cliente chamado. Este cliente pode estar inicializado e aceitar a sessão, pode não aceitar a sessão e pode até mesmo não estar inicializado.

  Parâmetro:
  - Identificador de Estado: indica o estado do cliente destino que será usado para informar o cliente origem.

- **Transmission.req**
  Direção: Usuário → KMPc

Descrição: Esta primitiva é invocada pelo cliente quando este deseja transmitir um conjunto de dados utilizando o serviço de criptografia.

Parâmetros:
- Identificador do cliente de origem: tupla formada pelo endereço IPv6 do nó onde está o cliente que solicita a sessão e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Identificador do cliente de destino: tupla formada pelo endereço IPv6 do nó onde está o cliente que irá participar da sessão juntamente com o cliente origem e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Dados: conjunto de informações a serem transmitidas utilizando o serviço de criptografia.

- **Transmission.ind**

Direção: KMPc → Usuário

Descrição: Esta primitiva informa o cliente que um conjunto de dados foi recebido pelo serviço de criptografia.

Parâmetros: retorna os parâmetros enviados em Transmission.req.

- **CloseSession.req**

Direção: Usuário → KMPc

Descrição: Esta primitiva é invocada pelo cliente quando este deseja fechar a sessão iniciada.

Parâmetros:
- Identificador do cliente de origem: tupla formada pelo endereço IPv6 do nó onde está o cliente que solicita a sessão e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1;
- Identificador do cliente de destino: Tupla formada pelo endereço IPv6 do nó onde está o cliente que irá participar da sessão juntamente com o cliente origem e o número do porto por este usado. O endereco IPv6 é um número inteiro entre 0 e $(2^{128} - 1)$ enquanto que o número do porto é um número inteiro entre 0 e 64K−1.

- **CloseSession.ind**

Direção: KMPc → Usuário

Descrição: Esta primitiva informa o cliente que um pedido de fechamento de sessão foi recebido pelo serviço de criptografia.

Parâmetros: retorna os parâmetros enviados em CloseSession.req.

2.3.2 Unidades de Dados do Protocolo (PDU's)

A seguir é apresentada a descrição detalhada de cada um das unidades de dados do protocolo (PDU's).

KMP_ID: mensagem de identificação enviada quando um nó da rede deseja cadastrar-se como um usuário da rede segura. Quando o Servidor de Chaves e Autenticação (AKS) recebe uma mensagem deste tipo, ele verifica se o usuário está na lista dos componentes da rede segura e em caso positivo executa as seguintes ações: cadastra o usuário armazenando seu identificador e chave pública, envia sua chave pública através de uma mensagem KMP_PAKS e publica o valor da função hash para verificação. Caso o cliente não esteja na lista de clientes seguros, o AKS envia uma mensagem KMP_ERROR;

KMP_PAKS: mensagem enviada pelo Servidor de Chaves (AKS) quando este recebe o pedido de cadastramento de um cliente comprovadamente seguro, ou seja, que está no cadastro feito manualmente pelo administrador de segurança. Esta mensagem é composta apenas pela chave pública do AKS. Quando o usuário recebe esta mensagem ele retira e armazena a chave pública do AKS;

KMP_REQ: mensagem enviada pelo usuário quando este já está cadastrado no AKS e deseja uma conexão segura com outro usuário. Esta mensagem é composta pelo identificador do nó origem e pelo identificador do nó destino. Quando o AKS recebe esta mensagem, analisa se os dois identificadores contidos na mensagem representam nós seguros e em caso positivo responde com uma mensagem do tipo KMP_TICKET. Caso um dos nós não pertença à rede segura, o AKS responde com uma mensagem do tipo KMP_ERROR;

KMP_TICKET: mensagem enviada pelo AKS quando este recebe um pedido de conexão (KMP_REQ) válido. Esta mensagem contém o identificador e chave pública de cada um dos clientes envolvidos mas uma chave de sessão inicial e uma marca de tempo, os dois últimos cifrados com as chaves públicas dos usuários envolvidos. Todos estes dados estão na mensagem KMP_TICKET assinados com a chave privada do AKS. Quando o nó que solicitou a conexão recebe este pacote, ele verifica se foi realmente enviado pelo AKS através da decifragem com a chave pública do servidor, e em caso positivo cria a chave de sessão definitiva e o vetor de inicialização, enviados para o nó destino através de uma mensagem KMP_QUERY. O cliente origem também envia KMP_TICKET para o cliente destino exatamente como recebido do AKS;

KMP_QUERY: mensagem enviada pelo nó de origem da conexão para o nó destino após o recebimento correto de KMP_TICKET. É formada pela chave de sessão cifrada com a chave pública do nó destino e pelo vetor de inicialização cifrado com a chave de sessão em modo ECB. O cliente destino responde com uma mensagem KMP_STATUS;

KMP_STATUS: mensagem enviada pelo nó destino quando este recebe a mensagem KMP_QUERY de um nó seguro juntamente com a mensagem KMP_TICKET tal qual foi enviada pelo AKS. Esta mensagem é composta pela marca de tempo e um campo de status, assinados com a chave privada do nó destino. Pelo campo de status o cliente origem saberá se o cliente destino está pronto para a sessão;

KMP_ERROR: mensagem enviada pelo usuário do serviço quando detectada a presença de alguma anormalidade. É composta pelo identificador do nó que a

enviou mais um descritor de erro.

KMP_DATA: esta mensagem é enviada contendo dados cifrados com a chave de sessão. Esta pode ser enviada por qualquer dos nós envolvidos na sessão. Quando um nó recebe esta mensagem, decifra os dados utilizando a chave de sessão.

KMP_END: esta mensagem é enviada pelo cliente quando este deseja terminar a sessão segura.

2.3.3 Codificação das Mensagens

Abaixo encontra-se a descrição do formato de cada uma das mensagens do protocolo KMP. Toda mensagem possui um valor numérico, encontrado no primeiro byte, que a diferencia das outras.

KMP_ID: 111 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_ID
IDa	18	Identificador do nó
Pa	92	Chave pública do nó

KMP_PAKS: 93 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_PAKS
Paks	92	Chave pública do AKS

KMP_REQ: 37 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_REQ
IDa	18	Identificador do nó origem
IDb	18	Identificador do nó destino

KMP_TICKET: 448 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_TICKET
IDa	18	Identificador do nó origem
Pa	92	Chave pública do nó origem
IDb	18	Identificador do nó destino
Pb	92	Chave pública do nó destino
Epa	64	Ksi + T cifrados com a chave pública Pa
Epb	64	Ksi + T cifrados com a chave pública Pb

Podemos ver que o tamanho da mensagem KMP_TICKET é maior que a soma dos componentes listados na tabela acima. Isto se deve ao fato de que, sempre que ciframos uma mensagem com uma chave RSA, seja pública ou privada, utilizando módulo de 512 bits (64 bytes), o bloco de saída tem 64 bytes. O dado a ser cifrado, no entanto, não deve ultrapassar 53 bytes, pois os bytes restantes são usados durante a cifragem para completar o dado de entrada. Desta forma, para assinar o Ticket (que teria originalmente 349 bytes), este é dividido em sete blocos de entrada (seis de 50 bytes e um de 49 bytes) que geram, cada um, uma saída de 64 bytes, levando aos 448 bytes da mensagem.

KMP_QUERY: 73 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_QUERY
Ks	64	Chave de sessão cifrada com a chave pública do nó destino
IV	8	Vetor de inicialização cifrado com a chave de sessão Ks

KMP_ERROR: 2 bytes

Campo	Bytes	Descrição
packet_type	1	tipo de mensagem = KMP_ERROR
Error_id	1	Identificador de erro

KMP_DATA: tamanho variável

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_DATA
data	variável	Dados cifrados com a chave de sessão

KMP_STATUS: 66 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_STATUS
Status	1	Resposta do cliente destino ao pedido de abertura de sessão
Sb(T)	64	Marca de tempo retirada do Ticket, assinada com a chave privada do cliente destino

KMP_END: 83 bytes

Campo	Bytes	Descrição
packet_type	1	Tipo de mensagem = KMP_END
Sa(Ida)	64	Chave pública do cliente de origem cifrada com sua chave secreta. Este campo será usado pelo cliente destino para autenticação da origem
IDb	18	Identificador do cliente de destino

2.3.4 Ambiente

A confiabilidade deste protocolo baseia-se na confiabilidade do ambiente onde é executado. Isto significa que do ponto de vista de sistema operacional nenhum outro processo pode ter acesso ao espaço de endereçamento da aplicação, ou pelo menos da sua área de dados. A comunicação é confiável e baseia-se no serviço oferecido pelo TCP.

2.3.5 Regras de Procedimento

O comportamento de clientes e servidores é descrito pelos autômatos representados nas figuras 4 e 5 respectivamente.

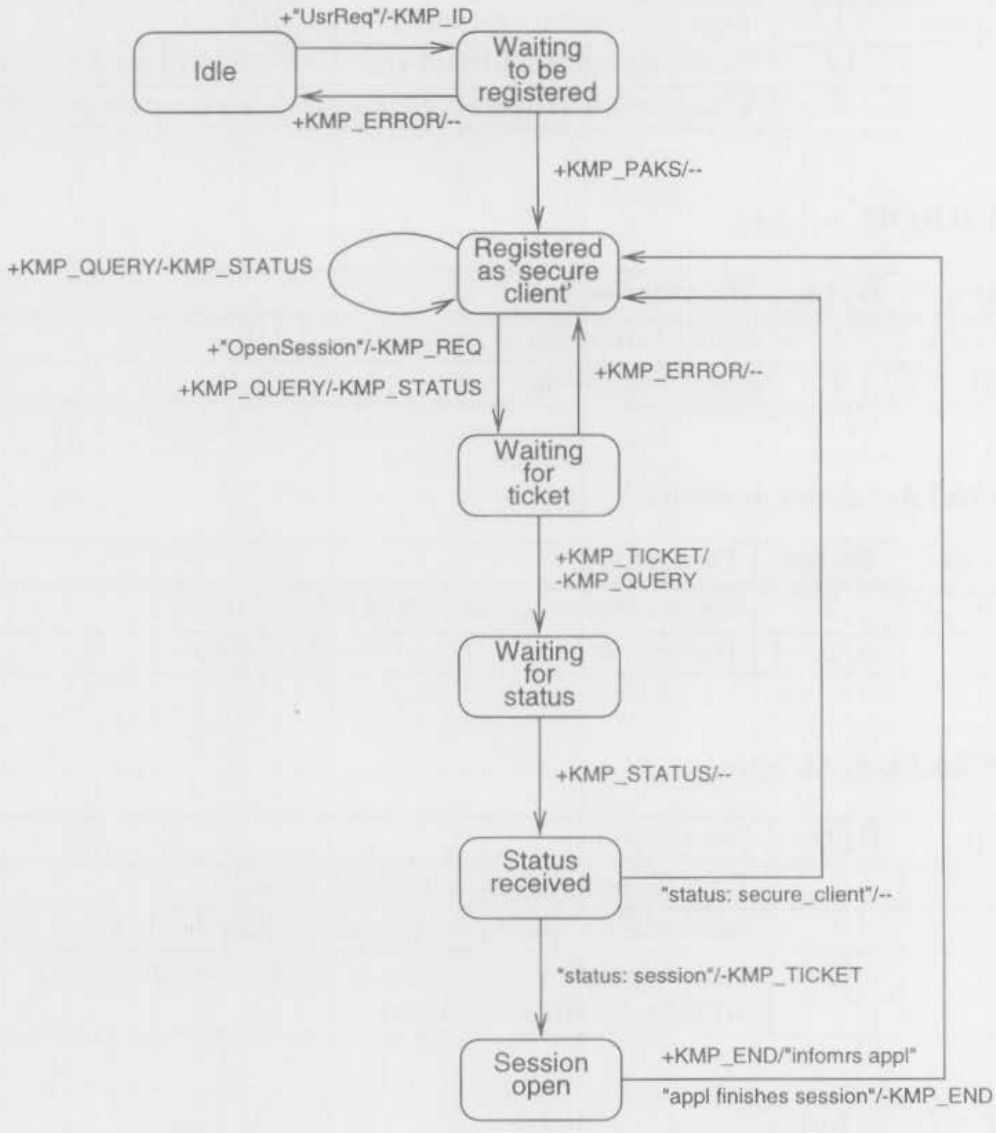

Figura 4: Autômato do KMP cliente

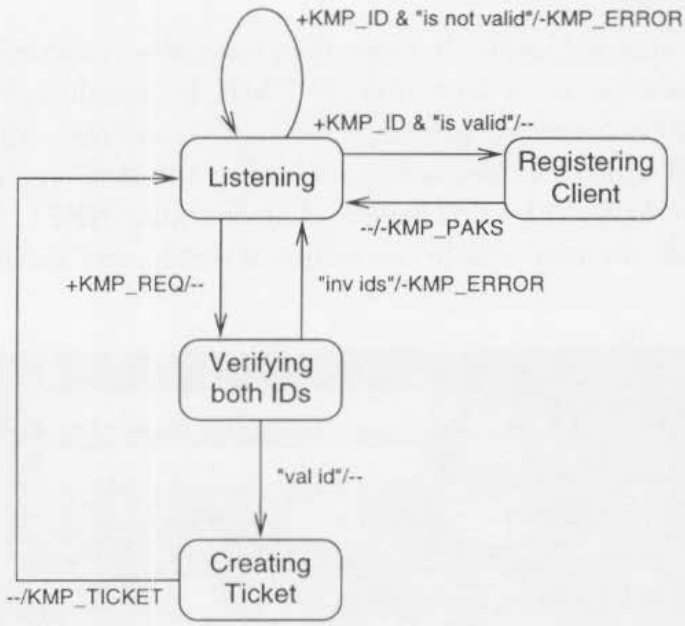

Figura 5: Autômato do KMP servidor

3 Implementação e Testes Executados

Para testar o protocolo proposto, o mesmo foi implementado no Laboratório de Redes de Alta Velocidade da Universidade Federal de Minas Gerais. O protocolo foi implementado em Java e construiu-se uma aplicação para utilizar seus serviços— um programa de troca de mensagens (talk). Para mostrar a viabilidade do uso do protocolo KMP foram feitos vários experimentos que têm como objetivo estimar o atraso (overhead) causado ao sistema pelo serviço de criptografia.

3.1 A escolha da Linguagem Java

A linguagem Java foi escolhida por um conjunto de fatores. Em primeiro lugar, Java é independente de plataforma, ou seja, o mesmo código construído e executado no ambiente Solaris pode ser executado sem uma única alteração em todas as plataformas para as quais existe um interpretador Java. Em segundo lugar, Java provê todas as ferramentas necessárias para a construção de uma aplicação gráfica, permite criar aplicações com várias threads de execução, provê mecanismos simples de envio de mensagens e permite que eventos sejam tratados de forma bastante intuitiva.

Um fator que foi importante na decisão da linguagem foi a obtenção do código dos algoritmos de criptografia utilizados (DES e RSA). A legislação dos Estados Unidos impede, por motivos de segurança, que implementações destes algoritmos sejam obtidas por pessoas fora do país. Na procura de uma implementação completa dos algoritmos de criptografia, encontrou-se um pacote desenvolvido em Java com estes algoritmos, entre outros. A Sun, através da Javasoft, possui um pacote com vários algoritmos de criptografia, mas as leis a impedem de adicioná-la ao Java Developer's Kit (JDK), utilizado neste trabalho em sua versão 1.1.4. O pacote utilizado chama-se IAIK-JCE e consiste em uma reimplementação do pacote da Sun por uma empresa austríaca. Este pacote foi encontrado por simples pesquisa na Internet.

3.2 A Aplicação

Foi implementada uma aplicação de troca de mensagens (talk) com interface gráfica que implementa todas as fases do protocolo KMP. Esta aplicação permite que dois usuários em máquinas remotas troquem mensagens seguras, cifradas com a chave de sessão gerada. A aplicação possui dois módulos: um implementando os clientes e outro com a função do Servidor de Chaves e Criptografia (AKS). Abaixo encontra-se a interface inicial do módulo cliente, antes que a sessão seja iniciada.

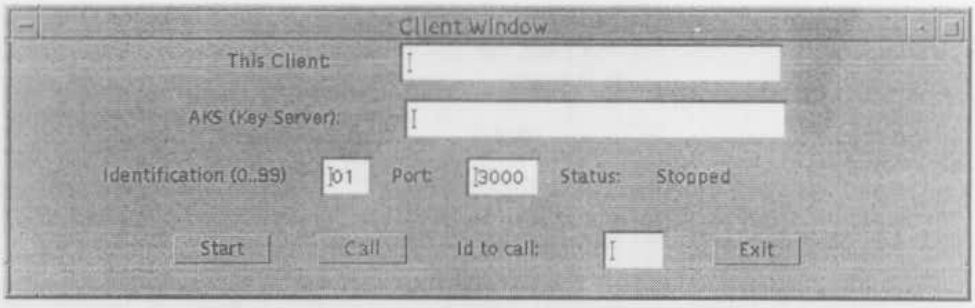

Figura 6: Interface cliente antes de iniciar aplicação

O usuário entra com o endereço IP da máquina onde se encontra o módulo AKS e a fase de identificação é iniciada através do botão "Start". Para dar início à sessão, informa-se o identificador do cliente destino e aciona-se o botão "Call". Após a troca de mensagens entre o cliente e o AKS os dois clientes possuem uma chave de sessão para ser utilizada na codificação dos dados. Um exemplo de execução pode ser visto na figura 7, quando a sessão já foi iniciada.

3.3 Testes

Para mostrar a viabilidade da implantação do protocolo, foram feitos uma série de testes. Abaixo temos uma descrição detalhada dos testes executados e algumas medidas de desempenho. A máquina utilizada para os testes foi uma Sun Ultra 1 com 64 MB de memória RAM. Quando os testes envolvem envio de mensagem, foi utilizada outra máquina para executar o AKS, uma Sun Ultra 1 com 48 MB de memória RAM. Em virtude da linguagem Java ser interpretada, pode-se ainda conseguir melhores resultados que os apresentados a seguir.

3.3.1 Tempo de Geração das Chaves Pública e Privada

Quando um cliente quer se cadastrar como cliente seguro, o serviço de criptografia vai executar primeiramente um código para gerar um par de chaves (pública e secreta) para este cliente. Após a geração deste par, vai contactar o AKS para o cadastramento propriamente dito.

Para medir o atraso criado pela criação do par de chaves criptográficas, foram feitas 1000 gerações aleatórias. Obteve-se um valor médio de 4439 ms para esta geração. Deve-se notar que tal procedimento é executado apenas na inicialização do cliente e não tornará a ser executado não importando o número de sessões que este venha a solicitar. Como dito anteriormente, este código é executado quando

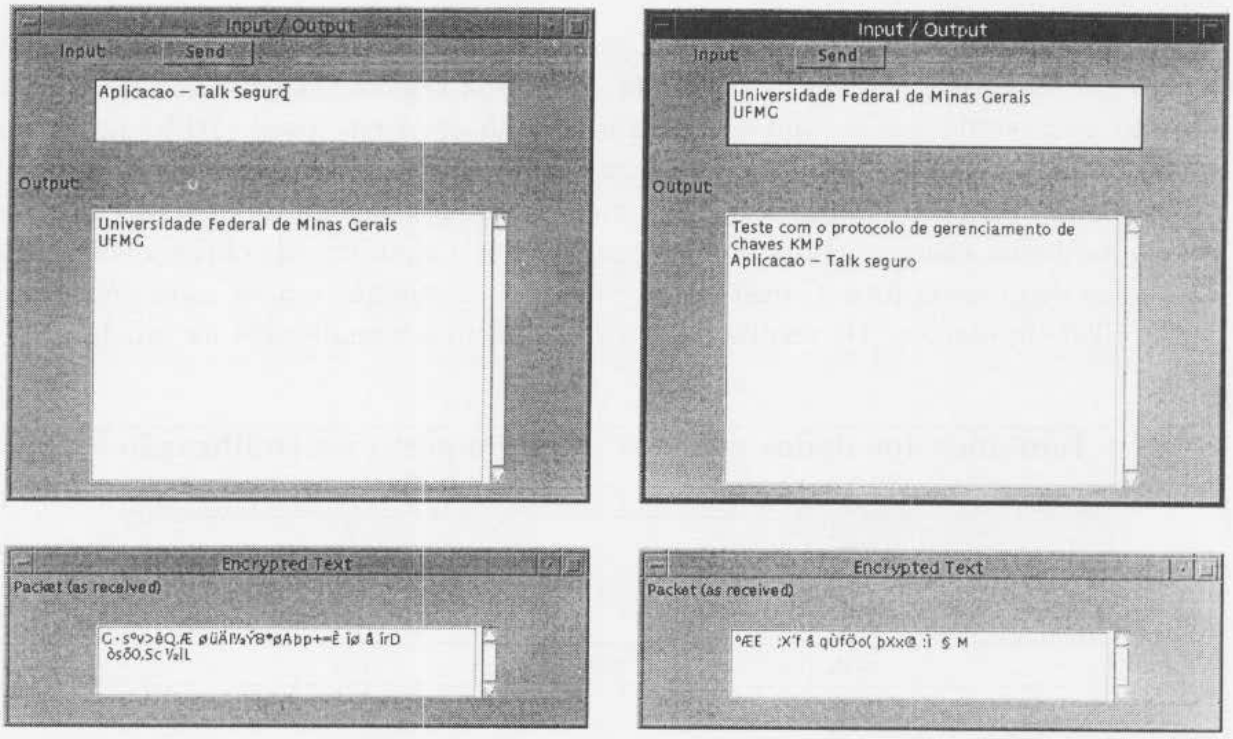

Figura 7: Exemplo de uma execução da aplicação

na inicialização de um novo cliente seguro ou quando este percebe que seu par de chaves está comprometido, não acarretando assim atraso considerável ao sistema.

3.3.2 Tempo de Identificação

Assim que o cliente tem seu par de chaves criado, o protocolo módulo KMPc vai entrar em contato com o AKS para pedir que o cliente seja cadastrado. Esta fase também gera um atraso e para verificar o tempo gasto com tal operação o AKS foi inicializado e um cliente fez 1000 requisições de cadastramento. Obteve-se um valor médio de 425 ms para as operações de (i) envio do pedido de cadastro, (ii) o cadastramento propriamente dito (feito pelo AKS) e (iii) a resposta do servidor, informando a atualização do cadastro.

3.3.3 Tempo de Abertura de Sessão

Uma vez cadastrado, um cliente está preparado para solicitar a abertura de sessões seguras. Quando desejar, o cliente envia um pedido de abertura de sessão (representado pela mensagem KMP_REQ) contendo seu identificador e o identificador do cliente com o qual quer se comunicar. O Servidor verifica se a sessão pode ser criada e em caso positivo envia o Ticket de autenticação para o cliente origem. Este, para criar a chave de sessão definitiva, deve assinar a chave de sessão inicial contida no Ticket com sua chave privada.

Para verificar o tempo gasto nesta operação um cliente seguro faz uma solicitação de sessão, espera uma resposta do AKS que, então, gera a chave de sessão. Este procedimento foi repetido em 1000 simulações e o tempo gasto nestes três passos foi, em média, de 474 ms.

3.3.4 Codificação usando a Chave de Sessão

Como dito anteriormente, uma vez que os dois clientes envolvidos na sessão saibam a chave a ser usada, podem enviar dados de forma segura. Os dados são cifrados usando a chave de sessão com o algoritmo de cifragem em bloco (DES) no modo CBC. Toda vez que uma mensagem é transmitida entre os clientes seguros, o KMP cifra os dados, gerando um atraso. Para medi-lo, geramos aleatoriamente a chave de sessão, os dados a serem criptografados e utilizamos o algoritmo de cifragem do DES. Os dados de entrada foram construídos em vários tamanhos e para cada um foram feitas 1000 simulações. Os resultados obtidos podem ser analisados na tabela 1.

Tamanho dos dados cifrados (bytes)	Tempo gasto na codificação (milissegundos)
256	7
512	15
1 K	31
5 K	152
10 K	309
64 K	1946

Tabela 1: Tempo de codificação dos dados

Pode-se notar que o atraso gerado na coficação dos dados é pequeno o que faz com que o protocolo seja adequado para ser utilizado em diferentes aplicações.

4 Trabalhos Relacionados

O protocolo de Needham e Schroeder [6] possui variantes que apresentam as características desejadas para uso em redes de computadores. Tanto o protocolo de Denning e Sacco quanto o protocolo Kerberos são exemplos dessas variações e serão discutidos nesta seção.

4.1 Kerberos

Kerberos [2] é um sistema de autenticação de propósito geral onde o usuário prova a sua identidade a um verificador (servidor), que decide se este deve ter acesso, baseado na sua identidade. Esse protocolo permite um esquema de controle de decisão arbitrariamente complexo ou tão simples quanto gravar a identificação da requisição.

O Kerberos é baseado em algoritmo de criptografia simétrico. Essa abordagem apresenta dois problemas. O número de chaves gerenciadas é muito grande pois o servidor precisa compartilhar uma chave única e secreta com cada entidade que faz parte da rede. Outro problema é com relação ao conhecimento da chave de sessão por parte do servidor, o que possibilita que ele faça o papel do usuário posteriormente.

O Kerberos ainda apresenta outras limitações. O Kerberos não é eficiente contra ataques que descobrem a senha do usuário (*password-guessing attacks*). Se um

usuário escolhe uma senha fácil, um atacante pode descobrir essa senha e se fazer passar pelo usuário. Outro problema é que o Kerberos precisa de um caminho confiável, através do qual as senhas são transmitidas. Se o usuário entra com a senha em um programa previamente modificado pelo atacante (*Trojan horse*) ou se o caminho entre o usuário e o programa de autenticação pode ser monitorado, o atacante pode obter informação suficiente para fazer o papel do usuário.

Outra desvantagem do Kerberos é a impossibilidade de automatizar a distribuição inicial das chaves com segurança. Isso ocorre devido à utilização de algoritmos simétricos para distribuição das chaves de sessão. Assim, para ser confiável, a distribuição de chaves mestras deve ser feita manualmente em toda a rede.

4.2 Denning e Sacco

O protocolo de Denning e Sacco [3] é um dos mais adequados para uso em redes de computadores, já que apresenta características como segurança, pequeno número de interações e máxima automatização da distribuição de chaves (menor quantidade de procedimentos manuais).

Para explicar o princípio de funcionamento desse protocolo usaremos a seguinte notação:

$$\begin{aligned} A &\rightarrow AKS : ID_A, ID_B \\ AKS &\rightarrow A : A_{S_K}(ID_A, P_A, T), A_{S_K}(ID_B, P_B, T) \\ A &\rightarrow B : E_{P_B}(A_{S_A}(K_S, T)), A_{S_K}(ID_A, P_A, T), A_{S_K}(ID_B, P_B, T) \end{aligned}$$

O AKS fornece as chaves públicas de A e de B junto com uma marca de tempo T, e a chave de sessão Ks é gerada por A. Um problema com este protocolo é que A pode se disfarçar de B para falar com um terceiro nó C, como apontado por Schneier [7], que propôs enviar os identificadores IDa e IDb junto com a chave Ks para resolver essa falha.

A geração aleatória da chave de sessão é feita pelo nó origem A e isto não é adequado em muitas situações. A geração de chaves com características aleatórias de forma prática e eficiente deve ser feita por algum dispositivo especial de hardware. Portanto, a geração deve ser centralizada no servidor.

5 Conclusões

O protocolo descrito neste trabalho é baseado no protocolo de Denning e Sacco [3], com algumas modificações fundamentais. As novidades importantes são a centralização da geração das chaves no servidor e o mecanismo de modificação da chave de sessão inicial gerada pelo servidor em uma chave de sessão desconhecida pelo servidor. Este é um ponto de segurança a mais que o protocolo oferece.

Algumas vantagens do protocolo KMP especificado e implementado neste trabalho em relação aos protocolos discutidos na seção 4 são:

- A distribuição de chaves públicas é automática, através da rede e só exige um procedimento manual—o de verificação das chaves públicas;

- A geração de chaves é centralizada e não compromete a confidencialidade em relação ao gerador da chave;
- O serviço de distribuição de chaves pode ser distribuído pela rede sem comprometer a segurança.

O fato de termos seguido os princípios de projeto de protocolos descritos por Holzmann [4] revelou vários pontos do protocolo KMP que não eram tratados ou estavam incompletos como apresentado em [8]. O passo seguinte agora é fazer a verificação da versão final do protocolo usando alguma ferramenta que aceite uma especificação em um Técnica de Descrição Formal [1] ou alguma outra ferramenta como Promela ou SPIN desenvolvidos pelo próprio Holzmann.

Os tempos apresentados na seção 3.3 para geração das chaves pública e privada, identificação, abertura de sessão e codificação mostram que é totalmente viável a utilização deste protocolo por uma aplicação. O trabalho a ser feito agora é a codificação dos algoritmos em uma outra linguagem como C e a disponibilização deste protocolo como um "toolkit" que possa ser facilmente incorporado a outras aplicações.

Referências

[1] S.T. Chanson, A.A.F. Loureiro, and S.T. Vuong. On Tools Supporting the Use of Formal Description Techniques in Protocol Development. *Computer Networks and ISDN Systems*, 25(7):723–739, February 1993.

[2] B. Clifford Newman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[3] D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communication of the ACM*, 24(8):533–536, August 1981.

[4] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ, USA, 1991.

[5] Antonio A.F. Loureiro and Osvaldo S.F. de Carvalho. On the design of communication protocols that support coordination loss. In *14º Simpósio Brasileiro de Redes de Computadores*, pages 553–573, Fortaleza, Ceará, Brasil, Maio 1996.

[6] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[7] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Willey & Sons, second edition edition, 1996.

[8] Dênio Teixeira Silva. Arquitetura de um sistema de segurança em redes atm com gerenciamento de chaves distribuído. Master's thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Março 1997.