Pequi: Uma Implementação de PKIX para Comunicações Seguras

Alexandre Reis e Silva
e-mail: arsilva@inf.puc-rio.br

PUC-RioInf.MCC08/99 Março, 1999

Abstract: Although the need for public key infrastructure has long been recognised for the widespread use of secure communication, the lack of easily available implementations has severely limited the use of public key cryptography to a small number of specific applications, with specialised solutions. Pequi is such an implementation designed to address a number of realistic wide-scale applications of this technology. The PKI implemented by Pequi is based on the integration of publicly available implementations of different components, to provide a network of certification authorities (CAs), which use LDAP for maintaining a distributed repository of certificates. Such implementation also performs automatic certificate path processing.

Keywords: communication security, public key infrastructure, certification paths.

Resumo: Embora a importância das infra-estruturas de chaves públicas em comunicações seguras seja amplamente reconhecida, a falta de implementações disponíveis no domínio público tem limitado severamente a utilização de criptografia de chaves públicas a um pequeno número de aplicações específicas, com soluções restritas. Pequi é uma implementação projetada para aplicações reais de larga escala dessa tecnologia. A PKI implementada pelo Pequi é baseada na integração de implementações de componentes distintos, disponíveis publicamente, para fornecer um rede de autoridades de certificação (CAs), as quais utilizam o LDAP para manter um repositório distribuído de certificados. Essa implementação também realiza a verificação automática de caminhos de certificação.

Palavras-chave: segurança de comunicações, infra-estruturas de chaves públicas, caminhos de certificação.

Índice

1. INTRODUÇÃO

Por possuirem reconhecida eficiência, as infra-estruturas de chaves públicas (*Public Key Infrastructures* - PKIs) vêm sendo cada vez mais empregadas em redes de computadores, fornecendo comunicações seguras por meio da utilização de certificados digitais. No entanto, as poucas implementações de domínio público existentes apresentam soluções apenas para casos muito específicos, sem oferecer a possibilidade de certificação entre hierarquias distintas e independentes.

Este projeto tem por finalidade a implementação de um protótipo compatível com as especificações da *Internet X.509 Public Key Infrastructure* (PKIX), padronização da *Internet Engineering Task Force* (IETF) para PKIs na Internet, que preencha essa lacuna, determinando dinamicamente e validando o caminho de certificação, que, por sua vez, pode ser constituído de certificados emitidos por autoridades de certificação (*Certification Authorities* - CAs) de diferentes hierarquias, armazenados em repositórios distribuídos.

O método de determinação/validação, desenvolvido para o protótipo, destina-se principalmente a PKIs de sistemas distribuídos em ambientes corporativos, possibilitando a comunicação entre entidades de diversas áreas de uma corporação, cada uma com sua PKI independente, ou até entre corporações distintas, através de certificados cruzados emitidos mutuamente, de comum acordo.

As seguintes interfaces de programação (APIs), descritas posteriormente em detalhes, são utilizadas na implementação:

- LDAP: operações de acesso a diretórios;
- Kerberos: autenticação forte para acesso a entradas dos diretórios LDAP;
- Oscar: manipulação de certificados e operação de PKIs.

São descritos, neste trabalho, os conceitos de repositórios distribuídos e caminhos de certificação, necessários à compreensão dos detalhes do projeto em pauta. Apresentam-se, em seguida, a especificação do sistema, seu projeto modular, roteiros de testes realizados, a documentação necessária à operação do sistema pelos usuários e, por fim, o código fonte do protótipo implementado.

2. CONCEITOS

2.1. Caminhos de certificação

O elemento básico de um caminho de certificação é o certificado digital. Trata-se de um conjunto de informações de uma entidade, incluindo a chave pública correspondente à sua chave privada, cuja confidencialidade é fundamental para sua autenticação. Esse conjunto de informações é assinado digitalmente por uma autoridade de certificação, a CA, na qual a entidade deve possuir confiança. CAs podem emitir certificados para outras CAs. A principal função do certificado é associar um nome (DN) a uma chave pública.

Para que uma entidade A possa autenticar outra, B, a primeira deve obter um conjunto de certificados que forme uma cadeia de confiança entre os dois pontos. Essa cadeia de confiança é chamada de caminho de certificação. A relação entre os certificados que compõem o caminho é: o proprietário do certificado corrente é o emissor do certificado imediatamente inferior, quando se percorre a hierarquia no sentido ascendente. O certificado seguinte tem como proprietário o emissor do certificado corrente, e assim sucessivamente.

A

Figura 1 ilustra uma hierarquia de certificação, destacando o caminho entre A e B. A notação utilizada para um certificado de Y, emitido por X, é X<<Y>>.

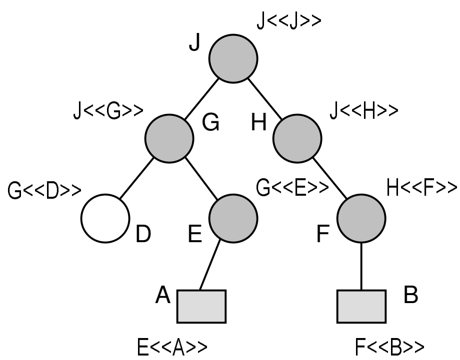

Figura 1 - Exemplo de caminho de certificação em uma hierarquia.

Na mesma Figura 1, os círculos representam as CAs e os retângulos as entidades usuárias da infra-estrutura de chaves públicas (PKI). A CA de mais alta hierarquia é chamada de CA raiz, cujo certificado é auto-assinado. Os círculos escuros destacam o caminho de certificação, quando A deseja autenticar B. O caminho é composto pela seguinte seqüência de certificados:

J<<J>> J<<H>> H<<F>> F<<B>> .

Como o certificado J<<J>> faz parte do conjunto de certificados mais confiáveis, que ligam A a sua CA raiz, pode-se realizar a validação do caminho de certificação a partir desse ponto. Se a entidade B possuir certificado emitido por D, ao invés de F, o caminho de certificação passa a ser:

G<<J>> G<<D>> D<<B>> .

Não há necessidade de verificação de certificados acima de G<<J>>, no caso logo acima, pois este faz parte do conjunto de certificados "mais confiáveis", composto por:

J<<J>> J<<G>> G<<E>> .

Maiores detalhes sobre PKIs, certificados e caminhos de certificação podem ser vistos em **[X509]** e **[PKIX]**.

2.2. Repositórios distribuídos

Repositórios são os componentes de infra-estruturas de chaves públicas responsáveis por armazenar e disponibilizar certificados e listas de certificados revogados (*Certificate Revocation Lists* - CRLs). Dependendo do tipo de utilização, podem-se implementá-los por meio de bancos de dados ou diretórios. Os diretórios são bancos de dados menos complexos, onde se armazenam informações lidas com freqüência e raramente alteradas.

Uma solução para o armazenamento de dados de PKI em repositórios, é o acesso ao repositório via *Lightweight Directory Access Protocol* (LDAP), protocolo simples, largamente utilizado na comunidade Internet e com implementações de domínio público. Trata-se de um protocolo pelo qual parceiros realizam transações relativas a qualquer diretório hierárquico, baseado em atributos.

Os dados que compõem a árvore de diretórios de informações (DIT - *Directory Information Tree*) LDAP podem ser armazenados em um ou mais servidores LDAP. Cada elemento da árvore é chamado de "entrada", que é identificada por um *Distinguished Name* (DN).

Simplificação do protocolo ITU-T X.500, o LDAP difere basicamente deste último por operar diretamente sobre TCP, não sendo necessária a implementação de todas as camadas OSI. Outra diferença é que o LDAP não realiza buscas completas no diretório. Ao invés disso, quando não forem achadas as informações requisitadas no servidor, o mesmo retorna ao cliente um *referral*, indicando onde tais informações podem ser encontradas. O cliente, quando recebe um *referral*, repete a operação em outro servidor LDAP. A Figura 2 ilustra o modelo de utilização do LDAP.

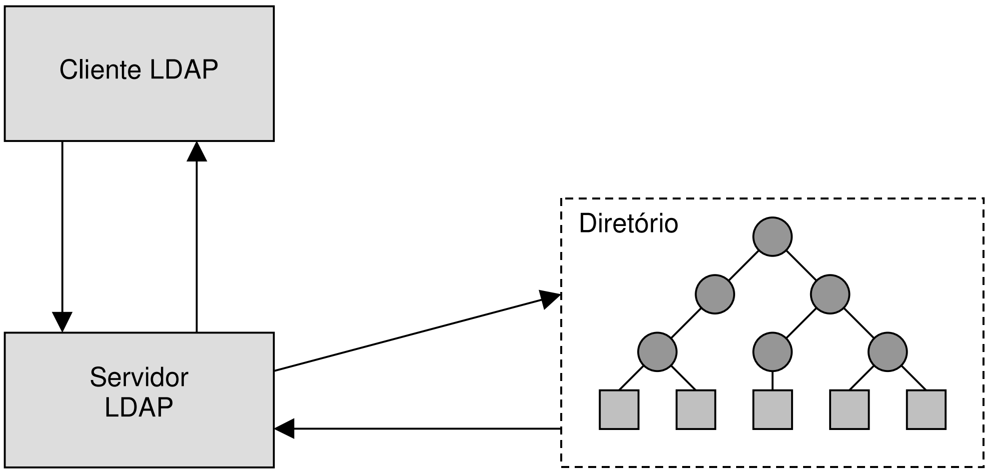

Figura 2 - Modelo de utilização do protocolo LDAP.

Por possuirem APIs bem estruturadas, boa portabilidade e por serem de utilização relativamente fácil [RFC1823], as implementações do LDAP existentes podem ser empregadas em aplicações que exijam facilidade de gerenciamento das informações armazenadas e escalabilidade. Alguns exemplos de utlização são: catálogos de usuários, repositórios de certificados/CRLs e administração de sistemas distribuídos. Nesta última aplicação, o LDAP permite que sejam armazenadas, apenas em uma entrada do diretório, todas informações de um usuário necessárias para que o mesmo acesse todo o sistema com apenas um *login*. A implantação/exclusão desse usuário no/do sistema é feita por meio de apenas uma operação de acesso ao diretório.

3. ESPECIFICAÇÃO DO SISTEMA

3.1. Objetivo

O presente projeto tem como objetivo implementar a verificação de certificados de usuários ou entidades, armazenados em repositórios distribuídos, realizada em duas etapas:

a) **determinação do caminho de certificação**: recuperação dos certificados que compõem o caminho entre o certificado que se deseja verificar e o certificado da entidade de maior confiança do usuário verificador, estabelecendo assim um caminho de confiança;

b) **validação do caminho de certificação**: verificação semântica das informações contidas em cada certificado, visando validar o caminho determinado na etapa anterior, de acordo com uma política de segurança pré-determinada.

O processo de determinação do caminho de certificação não é objeto de padronização do PKIX. Em conseqüência, fica a cargo do programador estudar a melhor maneira de implementá-lo [PKIX].

Quanto à utilização do LDAP como interface para repositórios, são definidos pelo PKIX o esquema de representação de informações [PKIXSch] e o protocolo para as operações de acesso ao repositório [PKIXOpp].

3.2. Requisitos

A autenticação de acesso a repositórios LDAP pode ser: nenhuma, senha em claro ou via Kerberos. Devido às necessidades de segurança da aplicação, para operações que alterem as informações do repositório, apenas a autenticação forte, por meio do Kerberos é permitida. Para operações de leitura, não há necessidade de autenticação, a não ser em casos específicos, onde alguns atribudos nas entradas de um diretório são sigilosos e outros não. A utilização de senha em claro nesses casos não é recomendada [PKIXOpp].

De modo a permitir a utilização das bibliotecas necessárias, a implementação do sistema foi realizada em ambiente Unix, mais especificamente, o Linux Red Hat 5.1. Como havia somente uma máquina com esse sistema operacional no Lab-DI, utilizou-se o Solaris 2.5.1 para possibilitar a utilização de algumas estações Sun como servidores LDAP em ambiente distribuído. Em uma dessas estações, foi também instalado um servidor Kerberos [KTH], para autenticação de acesso aos diretórios acessados via LDAP.

Para instalar uma PKI foram utilizadas funções de gerenciamento da PKI (tais como inclusão e exclusão de certificados em repositórios, geração de pares de chaves assimétricas e certificados), cujas implementações já eram disponibilizadas nos pacotes de distribuição das bibliotecas utilizadas:

a) **LDAP** (acesso a repositório)
   - ldapmodify: inclusão e alteração entradas e atributos no diretório;
   - ldapdelete: exclusão de entradas;

b) **Kerberos** (autenticação de acesso)
   - kinit: solicitação de um *Ticket-Granting Ticket* (TGT) ao servidor Kerberos;
   - klist: listagem de todos os TGTs válidos do usuário;

c) **Oscar** (PKI)
   - genkey: geração de par de chaves (RSA, DSA ou DH);
   - gencert: geração de certificados;
   - dumpcert: listagem de certificados em forma de texto;
   - certpath: validação de caminhos de certificação para testes, recuperando-se os certificados de arquivos locais;

Com o apoio das ferramentas citadas acima, foi determinada a implementação das seguintes operações do sistema:

a) **busca em diretórios distribuídos**: dado um filtro, um escopo e uma base, a partir da qual se inicia a busca, recuperam-se todas as entradas, sob essa base, que estejam em conformidade com o filtro, dentro do escopo especificado;

b) **leitura de entradas em diretórios distribuídos**: (simplificação da operação de busca) recupera atributos e valores de uma entrada armazenada em diretório distribuído;

c) **verifica caminho de certificação**: dado o DN de uma entrada, utiliza a operação de leitura para recuperar os certificados que formam o caminho de certificação e valida o caminho determinado.

3.3. Diagrama de especificação

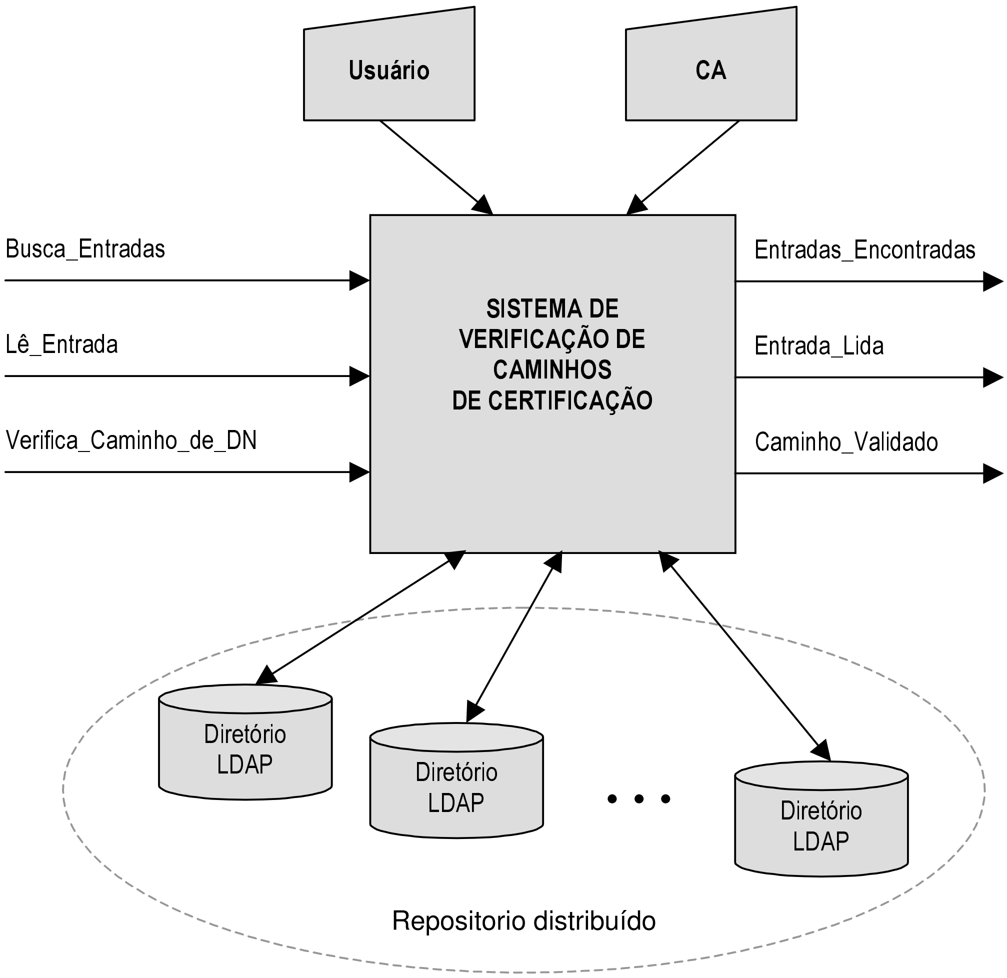

Figura 3 - Diagrama de especificação do projeto.

4. PROJETO MODULAR DO PROGRAMA

4.1. Bibliotecas utilizadas

4.1.1. Kerberos

4.1.1.1. Descrição

Parte integrante do KTH-Krb*, implementação sueca do protocolo Kerberos v4, essa interface de programação (API) permite a implementação de clientes e servidores, de maneira que possam autenticar-se mutuamente de maneira segura, por meio da utilização de tíquetes fornecidos pelo servidor Kerberos. O pacote KTH-Krb disponibiliza algumas implementações cliente-servidor, tais como: telnet, ftp, popper, rsh, login etc. São também fornecidas ferramentas para gerenciamento do serviço de autenticação em pauta e seus usuários.

4.1.1.2. Características utilizadas

O presente projeto não utiliza diretamente nenhuma característica dessa biblioteca. Contudo, para que seja possível a autenticação forte para acesso aos diretórios, as bibliotecas do Kerberos (libkrb.a e libdes.a) são necessárias para a compilação do LDAP.

Cabe ressaltar que o código fonte do KTH-Krb deve ser alterado, para que possam utilizar as portas TCP acima de 1023 e, portanto, sem restrições de uso. A alteração de numeração não é dada como opção, ao se compilar a biblioteca. A porta "padrão" do Kerberos é a 750. Para o presente projeto, utiliza-se a porta 11750.

4.1.2. LDAP

4.1.2.1. Descrição

Sua API define interfaces síncronas e assíncronas para acesso a diretórios, conforme [RFC1823] e [Howes], além de ferramentas para manipulação de atributos e valores de entradas. O acesso a diretórios, pela API do protocolo LDAP, se dá em quatro fases:

a) estabelecimento de conexão entre o cliente e um servidor LDAP;
b) autenticação perante o servidor LDAP;
c) operações de acesso - solicitação e obtenção de resultados;
d) encerramento da conexão.

A biblioteca escolhida para acesso a diretórios é OpenLDAP 2.0.1[†], de domínio público e baseada na implementação da Universidade de Michigan, LDAP 3.3[‡]. A opção pelo OpenLDAP deve-se ao fato de possuir maior portabilidade que o LDAP 3.3. Essa última não está mais sendo desenvolvida.

4.1.2.2. Características utilizadas

a) Interface: síncrona.

b) Estabelecimento de conexão

É a inicialização da interface. Consiste em abrir-se uma conexão com o servidor, dando-se como parâmetros um nome no formato DNS, ou um endereço IP, além do número da porta TCP onde se deseja conectar. A porta padronizada é a 389. Como as portas até 1023 são protegidas, optou-se pela 12345.

c) Autenticação

Permite controlar o acesso às informações do diretório. Pode ser nenhuma, para leitura e fraca (senha em claro) ou forte (Kerberos), para as outras operações. Como já dito anteriormente, por motivos de segurança, deve-se inibir o uso de autenticação fraca em operações que impliquem alteração das inforações nas entradas do diretório.

d) Busca

[*] KTH - Real Instituto de Tecnologia da Suécia. http://www.pdc.kth.se/kth-krb.
[†] Maiores informações podem ser obtidas em http://www.OpenLDAP.org .
[‡] Nessa implementação do protocolo LDAP baseiam-se todas as outras existentes atualmente. Mais informações em http://www.umich.edu/~dirsvcs/ldap.

É responsável pela obtenção de entradas do diretório, dados a base de busca, seu escopo, um filtro e os atributos que se deseja visualizar. Para a operação de leitura, restringe-se essa busca a uma só entrada.

e) Encerramento da conexão

Encerra a conexão estabelecida pela operação descrita no item b, acima.

f) Manipulação de DNs

Permite a separação de *Distinguishe Names* (DNs) em *Relative Distinguishe Names* (RDNs), para que se possa validar cadeias de caracteres como DNs e verificar se dois DNs estão relacionados hierarquicamente.

g) Manipulação de URLs

Destina-se a colher informações da cadeia de caracteres fornecida como *referral*. Dessa maneira, determina-se onde serão realizadas as próximas buscas/leituras.

h) Manipulação de valores binários

Possibilita a manipulação, armazenamento e recuperação de fotos jpeg, certificados, CRLs e qualquer outro tipo de valor binário.

4.1.3. Oscar

4.1.3.1. Descrição

Desenvolvida em C++ no *Distributed Systems Technology Center* (DSTC), da Universidade de Queensland, Austrália, essa biblioteca, em sua versão 3.0b4, oferece uma vasta gama de funções necessárias ao gerenciamento de PKIs e à manipulação de certificados, todos compatíveis com a padronização do PKIX.

O Oscar engloba outras bibliotecas, tais como: SNACC, para codificação e decodificação ASN.1; RSA, para criptografia assimétrica; e a implementação de criptografia simétrica DES da biblioteca SSLeay, também desenvolvida na Austrália. Estão também disponíveis alguns dos protocolos da série PKCS.

4.1.3.2. Características utilizadas

a) Codificação/decodificação DER

Os certificados são codificados em DER (subconjunto do ASN.1), para que possam ser armazenados nos repositórios. Esta característica permite a conversão desse formato para binário e vice-versa. A manipulação do certificado é feita no formato binário.

b) Armazenamento/recuperação de arquivos

Permite o armazenamento e a recuperação de dados binários, ou codificados em DER, em/de arquivos de maneira transparente ao programador.

c) Recuperação de DNs de proprietário e emissor

Recupera de um certificado o DN de seu proprietário ou emissor.

d) Verificação da extensão keyUsage

Possibilita escolher-se um certificado, para uma determinada finalidade, a partir de uma lista de certificados de um mesmo proprietário.

e) Caminho de certificação

Dada uma lista de certificados, verifica vários aspectos semânticos com o objetivo de validar o caminho. São verificadas várias extensões desses certificados, tais como basicConstraints, nameConstraints, certPolicies, policyMappings, policyConstraints, dentre outras.

f) Verificação sintática do certificado

Verifica a integridade de um certificado, utilizando-se a chave pública de seu emissor.

4.2. Política de segurança utilizada

A política definida abaixo, aplicável a sistemas distribuídos em ambientes corporativos e inter-corporativos, define os critérios utilizados para a definição do projeto em pauta:

a) uma CA somente emite certificados para outras CAs ou usuários a ela subordinados;

b) apenas a CA raiz tem a possibilidade de emitir certificados para CAs raízes de outras hierarquias (certificação cruzada);

c) CAs em uma mesma hierarquia não devem realizar certificação cruzada, seja ela vertical, ou horizontal (entre CAs de mesmo nível hierárquico);

d) não há transitividade de confiança, em relação a outras hierarquias, ou seja, se X confia em Y e Y, em Z, não implica que X deva confiar em Z, por intermédio de Y;

e) podem-se utilizar múltiplos certificados para cada entrada, destinados a aplicações distintas;

4.3. Descrição dos módulos do sistema

Os seis módulos do presente sistema serão descritos a seguir:

4.3.1. Módulos principais

a) search.c

Responsável pelas operações de busca e leitura em diretórios LDAP distribuídos. Fornece os atributos e respectivos valores para cada entrada encontrada.

Utiliza as bibliotecas Kerberos e LDAP. Na operação de leitura, podem-se armazenar os valores binários encontrados em arquivos.

b) path.cc

Dado um DN, determina o caminho de certificação que liga esse DN ao certificado da entidade de maior confiança do usuário verificador (sua CA raiz). O acesso desse módulo ao repositório se dá pelo módulo search.c. Utiliza a biblioteca do Oscar para validar o caminho encontrado. Pode escolher, dentre um conjunto de certificados, aquele cujo uso seja permitido para uma determinada aplicação. Neste módulo, são utilizadas também as bibliotecas Kerberos, LDAP.

4.3.2. Módulos de search.c

a) io.c

Responsável pela impressão na tela dos atributos e valores encontrados de cada entrada, bem como pela gravação de dados binários em arquivos.

b) bmatch.c

Verifica se uma base de busca faz parte da hierarquia corrente.

c) referral.c

Separa as URLs da informação de *referral*, fornecida pelo servidor LDAP, e separa dessas strings de URL as informações necessárias a novas buscas em outros servidores, como DNS ou endereço IP do servidor, porta e base de busca.

d) visit.c

Verifica se uma base de busca foi visitada, para evitar *referrals* que indiquem o servidor onde se iniciou a busca e formem um ciclo, causando um *loop* contínuo na execução do programa.

4.4. Diagrama da arquitetura de módulos do sistema

A Figura 4 abaixo ilustra a relação entre os módulos do sistema.

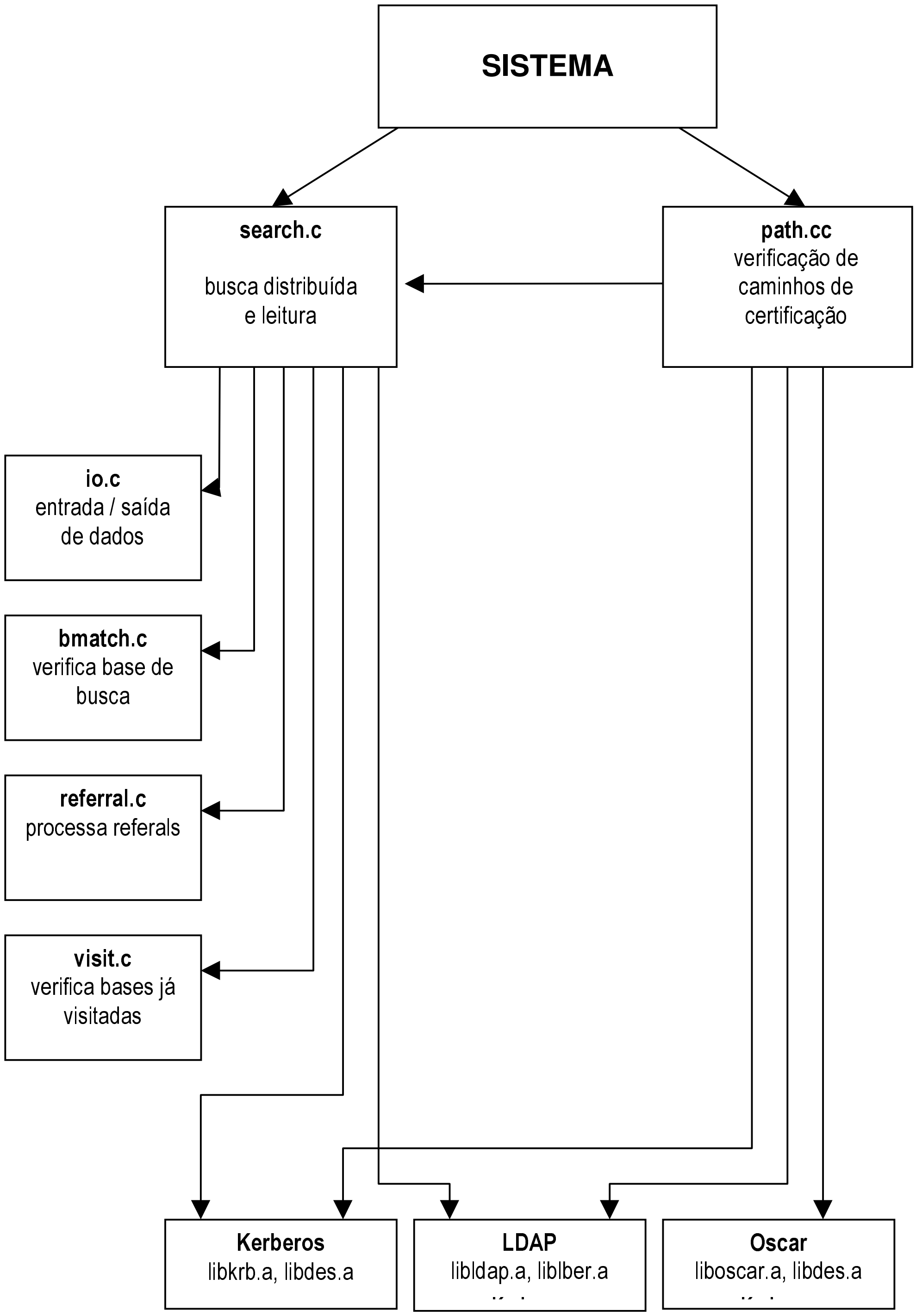

Figura 4 - Diagrama de módulos do projeto.

4.5. Diagrama da seqüência de operações para verificação de um caminho

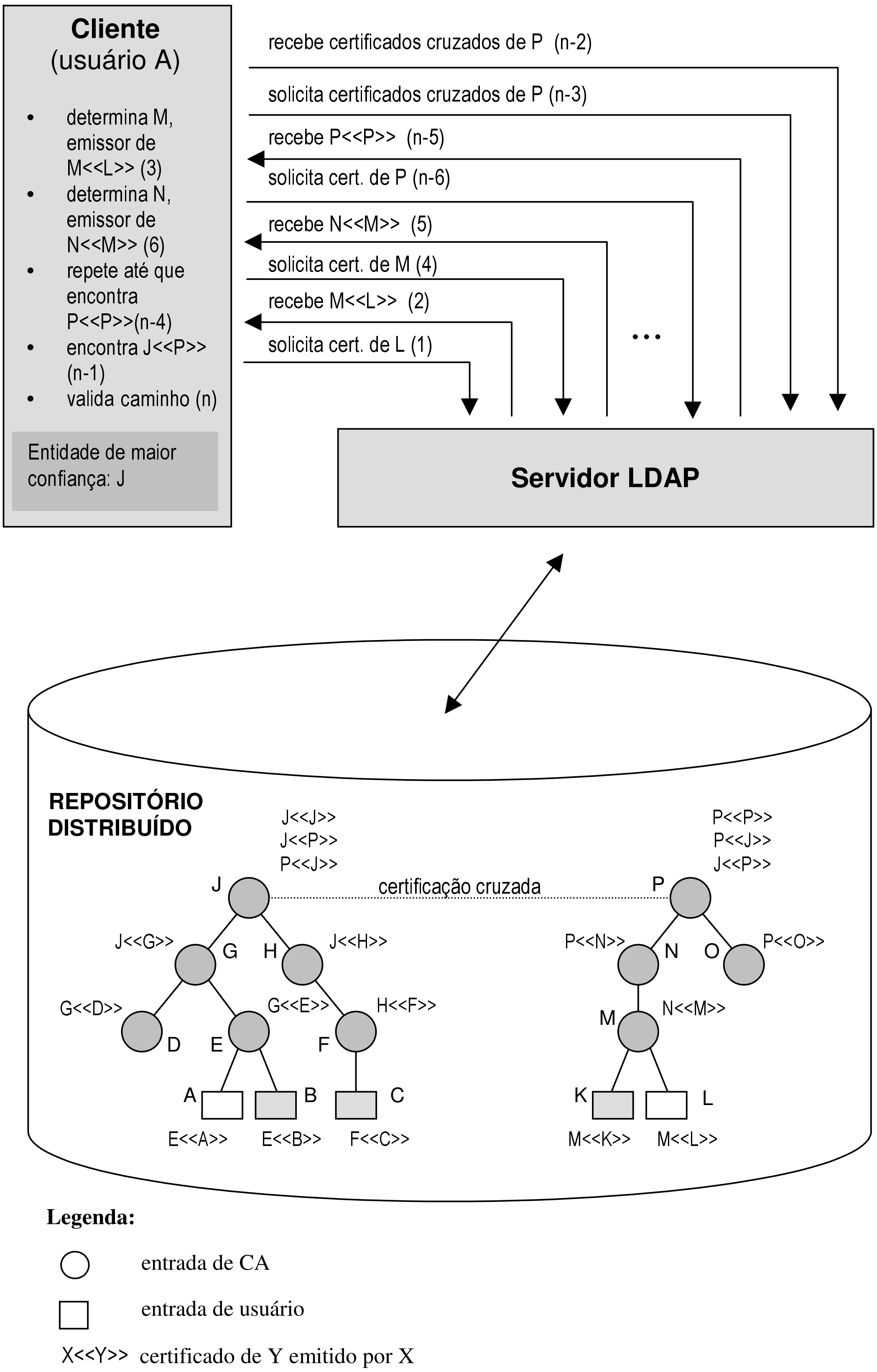

Figura 5 - Diagrama da seqüência de operações de verificação de caminhos.

4.6. Linguagens de programação utilizadas

Por ter sido intenção inicial utilizar como biblioteca para PKI o SSLeay*, desenvolvido na Austrália em C, e por serem as outras APIs também desenvolvidas na mesma linguagem, optou-se pela implementação do projeto em C, sob o sistema Solaris 2.5.1 do Lab-DI, pela disponibilidade de estações de trabalho para a realização de testes em ambientes distribuídos.

Somente após o término da implementação da operação de busca é que o Oscar, desenvolvido em C++ para ambientes Solaris 2.6 e Linux Red Hat 5.1, tornou-se público. Devido a este último possuir uma API bem mais completa que o SSLeay, em termos de manipulação de certificados, eliminando a necessidade de implementar-se a codificação/decodificação ASN.1, tarefa essa que consumiria em torno de quatro meses de trabalho, optou-se por utilizá-lo na implementação da operação de verificação de caminhos de certificação. Em conseqüência, migrou-se o código já implementado para o ambiente Linux e a última etapa do projeto foi desenvolvida em C++, mantendo-se contudo a lógica procedural planejada.

4.7. Descrição das funções do sistema

a) Módulo path.cc

```
int PKIPATH_tpPath( char*            dn ,
                    PKISRCH_tpfFlag fNeedKeyUsage ,
                    KeyUsageType    keyUsage        )
```

Realiza a verificação do caminho de certificação que liga dn ao usuário verificador.

Valores de entrada:

dn - *Distinguished Name* do usuário (ou da entidade) a ser verificado;

fNeedKeyUsage - flag que indica se há necessidade de verificação da extensão keyUsage de cada certificado do caminho;

keyUsage - especifica o tipo de utilização à qual se destina o certificado do usuário.

---

* Mais informações podem ser obtidas em http://psy.uq.oz.au/~ftp/Crypto.

Valores de retorno:

0 - verificação de caminho bem-sucedida;

-1 - erro na verificação (uma mensagem de erro é impressa na tela).

b) Módulo search.c

```
int PKISRCH_tpSearch( PKISRCH_tpSearchArgs*    args )
```

É responsável pelas operações de busca e leitura de entradas em repositórios distribuídos. Os valores e atributos das entradas encontradas são impressos na tela. Esta função é recursiva, permitindo a busca automática a partir de *referrals*.

Valores de entrada:

args - estrutura que contém todos os dados necessários à operação, tais como DNS ou endereço IP do servidor LDAP, porta TCP, base de busca (DN do usuário a se verificar), filtro, escopo etc.

Valores de retorno:

0 - busca ok;

-1 - erro na busca (mensagem de erro na tela).

c) Módulo io.c

```
void PKISRCH_tpOutputEntry( LDAP*             ld ,
                            LDAPMessage*      entry ,
                            PKISRCH_tpfFlag   fBinToFile )
```

Processa cada entrada encontrada, imprime na tela seus atributos e valores, bem como armazena valores binários em arquivos.

Valores de entrada:

ld - *handle* contendo as informações sobre a conexão LDAP;

entry - contém todas as informações pertencentes à entrada;

fBinToFile - determina se há necesidade armazenamento de valores binários em arquivos.

Valor de retorno:

Nenhum. Apenas emite mensagem de erro se não for possível armazenar algum valor em arquivo.

d) Módulo bmatch.c

```
int PKISRCH_tpBaseMatch( char*    newbase ,
                         char*    parentbase )
```

Determina se uma base é igual ou subordinada à uma base de referência.

Valores de entrada:

newbase - DN da base a ser verificada;

parentbase - DN da base de referência.

Os DNs dos dois valores de entrada devem sempre possuir um espaço após cada vírgula. Não há distinção entre caracteres maíusculos e minúsculos.

Valores de retorno:

1 - base é subordinada à de referência;

0 - base é a de referência;

-1 - base não é subordinada à de referência.

e) Módulo referral.c

```
int PKISRCH_tpGetUrl( char*     referralString )
```

Separa as cadeias de URLs encontradas na cadeia de *referrals*, retornada pelo servidor LDAP. As URLs encontradas são armazenadas em um vetor de cadeias de URLs, para posterior manipulação.

Valor de entrada:

referralString - cadeia de caracteres contendo informação sobre os *referrals* encontrados em operações de busca ou leitura.

Valores de retorno:

1 - processamento bem-sucedido;

0 - impossível extrair URLs da cadeia de entrada (mensagem de erro na tela).

```
int PKISRCH_tpParseReferral( char*  url )
```

Extrai das URLs, definidas pela função anterior, o DNS do servidor, a porta para conexão e a nova base de busca. Essas informações são armazenadas numa estrutura do LDAP, do tipo LDAPURLDesc, comum aos módulos, para serem utilizadas pela função tpSearch, permitindo iniciar novas buscas/leituras.

Valor de entrada:

url - cadeia de caracteres contendo uma URL.

Valores de retorno:

0 - processamento bem-sucedido;

1 - cadeia em formato diferente de URL ou incompatível com URLs LDAP (mensagem de erro na tela).

f) Módulo visit.c

```
DDSRCH_tpfFlag PKISRCH_tpVisited(  char*   newbase )
```

Informa se uma base de busca já voi visitada.

Valor de entrada:

newbase - DN da base a ser verificada.

Valores de retorno:

YES - base já visitada;

NO - base ainda não visitada.

4.8. Descrição das estruturas de dados

Serão descritas abaixo as estruturas de dados definidas no projeto em pauta:

a) search.h

`PKISRCH_tpAuthMethod`

– Descrição: método de autenticação ao servidor LDAP.
– Tipo: enum
– Valores possíveis:

  NO_AUTH - nenhuma autenticação;

  KRB_AUTH - autenticação via Kerberos.

`PKISRCH_tpfFlag`

– Descrição: flag genérica.
– Tipo: enum
– Valores possíveis:

  NO - nenhuma autenticação;

  YES - autenticação via Kerberos.

`PKISRCH_tpSearchArgs`

– Descrição: armazena todos os argumentos necessários à busca/leitura em diretórios distribuídos; devido à recorrência das funções de busca e leitura, é necessário o armazenamento de alguns valores iniciais para futura consulta; os elementos iniciados por "original" têm esta incumbência.
– Tipo: struct

– Elementos da estrutura:

char**	attrs	- atributos a recuperar do diretório;
char*	user	- usuário para efeito de autenticação;
char*	originalfilter	- filtro da busca inicial;
char*	filter	- filtro corrente;
char*	originalbase	- base da busca inicial;
char*	parentbase	- base de referência da busca corrente;
char*	base	- base da busca corrente;
char*	host	- servidor LDAP;
int	port	- porta de conexão com servidor LDAP;
int	originalscope	- escopo da busca inicial;
int	scope	- escopo da busca corrente;

PKISRCH_tpAuthMethod	auth	- método de autenticação;
PKISRCH_tpfFlag	fFollowReferral	- indica se a busca é local ou distribuída;
PKISRCH_tpfFlag	fBinariesToFiles	- indica se os dados binários devem ser gravados em arquivos.

b) read.h

`PKIREAD_tpPKIDataType`

– Descrição: informa o tipo de dados binários, em relação à PKI.

– Tipo: enum

– Valores possíveis:

NONE	- dados não relacionados à PKI;
UCERT	- certificado de usuário;
CACERT	- certificado de CA;
XCERTP	- par de certificados cruzados;
ALIST	- lista de revogação de autoridades (ainda não utilizado);
CLIST	- lista de revogação de certificados (ainda não utilizado);

PKIREAD_tpPKIValuesFound

- Descrição: armazena os tipos de cadeias de dados binários PKI encontrados e seus respectivos comprimentos.
- Tipo: struct
- Elementos da estrutura:

PKIREAD_tpPKIDataType	type	- tipo de dados PKI;
long int	len	- comprimento da cadeia.

5. TESTES EFETUADOS

Para se verificar a corretude de funcionamento dos módulos responsáveis pelas operações de busca e leitura, foram realizados testes em todas as posições das árvores componentes das hierarquias ilustradas nas Figura 6, Figura 7 e Figura 8 abaixo.

Os testes de verificação de caminhos de certificação foram realizados utilizando-se a infra-estrutura mostrada na Figura 8. Foram armazenados, nas entradas dos dois repositórios distintos, certificados com valores de extensões basicConstraints e keyUsage bem diversificados, de maneira a possibilitar a corretude da implementação. Os testes transcorreram de maneira satisfatória.

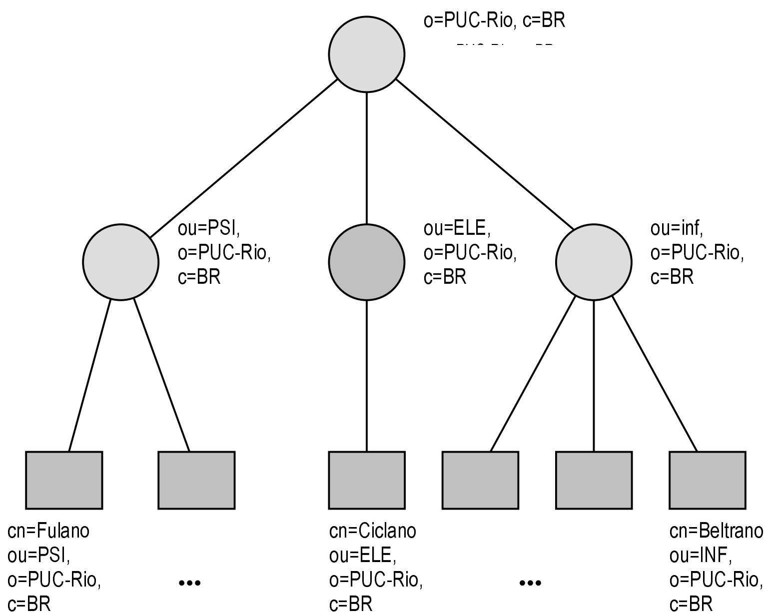

Figura 6 - Hierarquia simples.

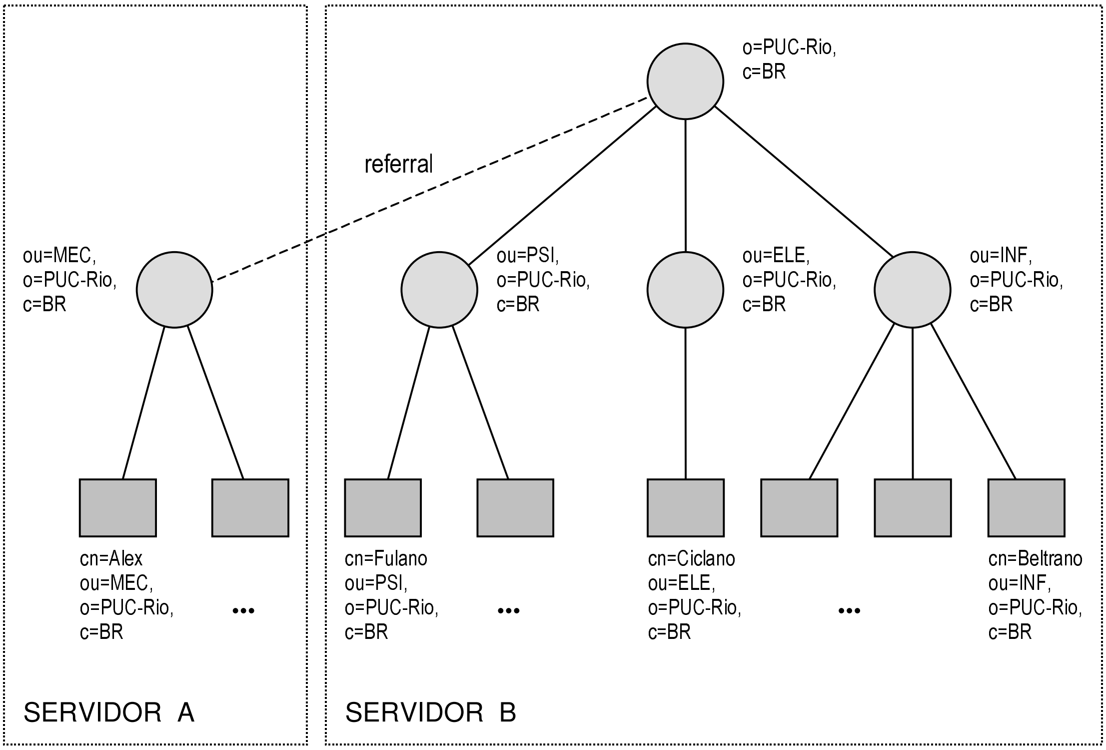

Figura 7 - Hierarquia simples distribuída em dois servidores.

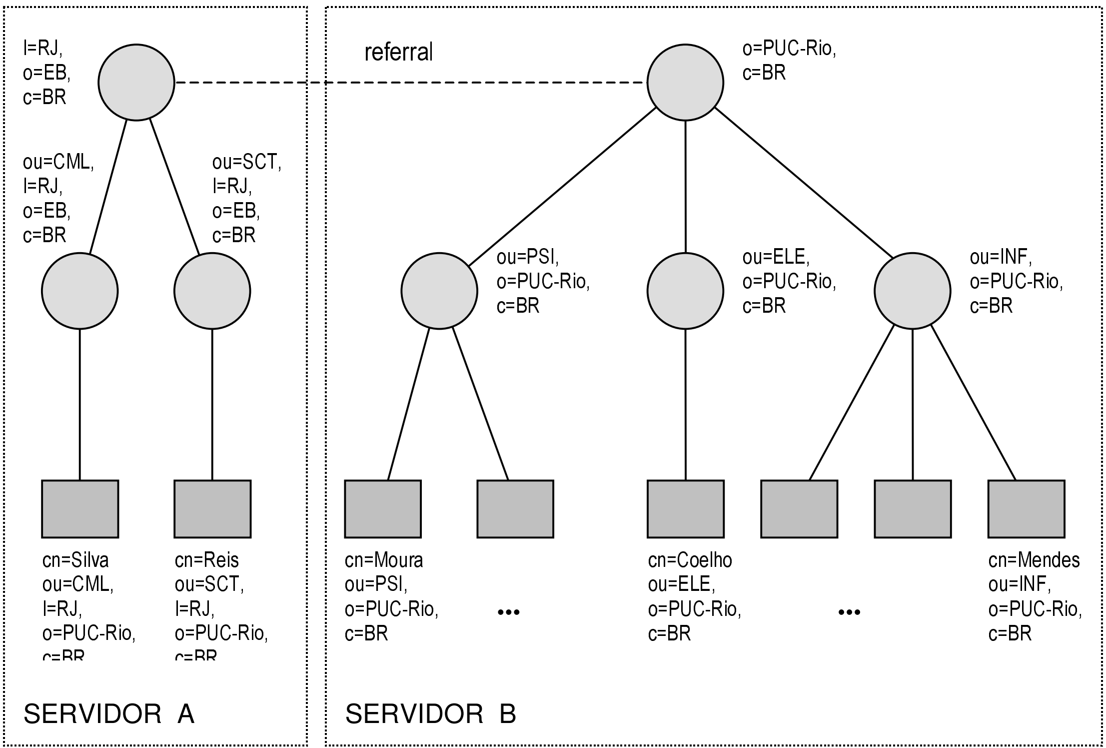

Figura 8 - Hierarquias distintas.

6. DOCUMENTAÇÃO PARA O USUÁRIO

6.1. Instalação

Para que as operações de busca, leitura e verificação de caminhos possam ser utilizadas, são disponibilizadas três rotinas principais para realizar a interface com o usuário. Antes da compilação, deve-se descompactar o pacote contendo os módulos e arquivos de cabeçalho:

```
tar xvzf pki.tar.gz
```

Cada uma das rotinas principais é compilada separadamente, conforme os comandos abaixo:

- **busca:** `make -f makesrch`
- **leitura*:** `make -f makeread`
- **caminhos:** `make -f makepath`

Além das bibliotecas mencionadas na seção 4.1 deste trabalho, a infra-estrutura necessária para compilação, em Linux Red Hat 5.1, é a seguinte†:

- compilador gnu C gcc-2.8.1;
- compilador gnu C++ g++-2.8.1;
- biblioteca de números de precisão arbitrária gmp-2.0.2;
- biblioteca padrão para C++ libstdc++-2.8.1.

6.2. Operação

Todas as rotinas são acessadas pela linha de comandos, bastando para isso digitar o nome do programa executável e os argumentos passados como suas opções, conforme a descrição abaixo:

---

* A compilação de search.c (e dos módulos por ele utilizados) para leitura é feita por meio da inserção do parâmetro -DREAD_ONLY, que é automaticamente inserido durante esse processo.
† Os pacotes necessários podem ser obtidos a partir de vários ponteiros de sites ftp existentes em http://www.redhat.com.

a) Busca em diretórios distribuídos

`pksearch [opções] filtro [atributos]`

opções:

`–r`		não segue os referrals encontrados
`–k`		autentica-se perante o servidor LDAP via Kerberos
`–b`	**`base`**	informa a base hierárquica onde se deseja iniciar a busca
`–s`	**scope**	informa o escopo da busca os valores de scope podem ser:
		`base` procura entrada referente à base de busca
		`one` procura um nível abaixo da base de busca
		`sub` procura toda a sub-árvore, a partir da base
-h	**`host`**	dns ou endereço ip do servidor para a busca inicial
-p	**`port`**	porta para conexão ao servidor LDAP
`–u`	**`user`**	dn do usuário para autenticação

`filtro`: filtro de busca (ex.: "(objectclass=person)", "(cn=alex*)")

`atributos`: atributos a serem recuperados de cada entrada (default: todos)

b) Leitura em diretórios distribuídos

`pkread [opções] filtro [atributos]`

`opções:`

`–r`		não segue os referrals encontrados
`–t`		armazena os valores binários encontrados em arquivos
`–k`		autentica-se perante o servidor LDAP via Kerberos
`–b`	**`base`**	informa a base hierárquica onde se deseja iniciar a busca
`–s`	**scope**	informa o escopo da busca os valores de scope podem ser:
		`base` procura entrada referente à base de busca
		`one` procura um nível abaixo da base de busca
		`sub` procura toda a sub-árvore, a partir da base
-h	**`host`**	dns ou endereço ip do servidor para a busca inicial

-p **`port`** porta para conexão ao servidor LDAP

`-u` **`user`** dn do usuário para autenticação

`filtro`: filtro de busca (ex.: "(objectclass=person)", "(cn=alex*)")

`atributos`: atributos a serem recuperados de cada entrada (default: todos)

c) Verificação de caminhos de certificação

`path [-n usage] dn`

`opção:`

`-n` **`usage`** permite escolher, dentre um conjunto de certificados de uma entrada, um que seja destinado a um determinado uso; os valores de *usage* podem ser:

- **0** assinatura digital
- **1** não-repúdio
- **2** encriptação de chaves
- **3** encriptação de dados
- **4** intercâmbio de chaves
- **5** assinatura de certificados
- **6** assinatura de CRLs (não implementada)
- **7** apenas encriptação
- **8** apenas decriptação

`dn`: indica o *distinguished name* do proprietário do certificado que se deseja verificar

Todas as informações dadas acima podem ser visualizadas na tela do computador, bastanto para isso digitar o nome do arquivo executável sem nenhum argumento.

7. SUGESTÕES PARA TRABALHOS FUTUROS

Este projeto aborda uma área controversa das infra-estruturas de chaves públicas*, que é a verificação de caminhos de certificação, possibilitando a interoperação de diversas estruturas hierárquicas independentes.

Apesar da complexidade do tema, trata-se apenas de uma parte de uma infra-estrutura. Para que se possa implementar na prática uma política de segurança, por meio de certificação, sugerem-se os seguintes temas, não constantes deste projeto:

– revogação de certificados, por meio de CRLs ou outro método que se achar mais eficiente, dependendo da aplicação;

– renovação de certificados, de maneira a evitar a indisponibilidade da infra-estrutura durante a transição, caso o certificado seja de uma CA raiz, por exemplo;

Em relação ao presente trabalho, as seguintes melhorias podem ser sugeridas:

– criação de uma cache para armazenar os caminhos verificados freqüentemente;
– armazenamento local, não só do certificado auto-assinado da CA raiz de maior confiança, como todos os certificados que formam o caminho até o certificado do usuário, visando economizar recursos de comunicação, caso se queiram verificar usuários que estejam na mesma árvore, sem necessidade de se recuperar todos os certificados até o da CA raíz;
– armazenamento da cache, do certificados do caminho de maior confiança, citados logo acima em um PSE (*Personal Secure Environment*), que é uma estrutura de dados armazenada em arquivos e criptografada;
– disponibilização do certificado do usuário autenticado para utilização por outras aplicações;
– implementação de CRLs.

* Consultar a lista de discussão do PKIX Working Group, do IETF, para maiores detalhes, em http://www.ietf.org.

8. BIBLIOGRAFIA

[Howes] **HOWES**, T., SMITH, M., "Programming Directory-Enabled Applications with Lightweight Directory Access Protocol", Macmillan Technology Publishing, 1997.

[KTH] **DANIELSSON**, J., WESTERLUND, A., "KTH-KRB (Kerberos 4 from KTH)", Edition 1.0 for version 0.9.8, KTH (*Kungliga Tekniska Högskolan* - Real Instituto de Tecnologia, Estocolmo, Suécia), dezembro de 1997, http://www.pdc.kth.se/kth-krb.

[PKIX] **HOUSLEY**, R., FORD, W., POLK, W., SOLO, D., "Internet Public Key Infrastructure - X.509 Certificate and CRL Profile", IETF Draft, PKIX Working Group, 11ª versão, setembro de 1998.

[PKIXOpp] **BOEYEN**, S., HOWES, T., RICHARD, P., "Internet X.509 Public Key Infrastructure - Operational Protocols - LDAPv2", IETF Draft, PKIX Working Group, 8ª versão, setembro de 1998.

[PKIXSch] **BOEYEN**, S., HOWES, T., RICHARD, P., "Internet X.509 Public Key Infrastructure - LDAPv2 Schema", IETF Draft, PKIX Working Group, 2ª versão, setembro de 1998.

[RFC1777] **YEONG**, W., HOWES, T., KILLE, S., "Lightweight Directory Access Protocol", RFC 1777, IETF, março de 1995.

[RFC1823] **HOWES**, T., SMITH, M., "LDAP Application Program Interface", RFC1823, IETF, agosto de 1995.

[X509] "Information Technology - Open Systems Interconnection - The Directory: Authentication Framework", ITU-T Recommendation X.509, 3ª versão, junho de 1997.

ANEXO A - CÓDIGO FONTE

Rotinas principais

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Certificate path processing (main program)
 *   Version:     1.0
 *   File:        mainpath.cc
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 25th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _MAINPATH_
#define _MAINPATH_

#include "path.hh"

/* Prints correct usage of this program */
void PKIPATH_tpPrintUsage( )
{
   cerr << "------------------------- PATH --------------------------"
<< endl
        << endl << "Usage: path [options] dn"                          << endl
        << "options:"                                                  << endl
        << "        -n keyusage    specify keyUsage extension needed:" << endl
        << "                           0 - digital signature"          << endl
        << "                           1 - non-repudiation"            << endl
        << "                           2 - key encipherment"           << endl
        << "                           3 - data encipherment"          << endl
        << "                           4 - key agreement"              << endl
        << "                           5 - key cert sign"              << endl
        << "                           6 - crl sign"                   << endl
        << "                           7 - encipher only"              << endl
        << "                           8 - decipher only"              << endl
        << "        dn             subject's dn to be verified "       << endl
        << "-------------------------------------------------------" << endl
        << "\a" ;
}

int main( int argc, char** argv )
{
   int              i ;
   char*            dn ;
   PKISRCH_tpfFlag  fNeedKeyUsage ;
   KeyUsageType     keyUsage ;

/*   dn            = DEFAULT_NAME ; */
```

```
   fNeedKeyUsage = NO ;
   keyUsage      = DEFAULT_USAGE ;

/* Parse command line */
   while ( ( i = getopt( argc, argv, "n:" ) ) != EOF )
   {
      switch ( i )
      {
         case 'n' :
         /* need keyUsage extension verification */
            fNeedKeyUsage = YES ;
            keyUsage      = (KeyUsageType) ( char* ) strdup( optarg ) ;
            break ;
         default :
            if ( argc > ONE )
            {
               PKIPATH_tpPrintUsage( ) ;
               return( NEG ) ;
            } /* if argc */
      } /* switch i */
   } /* while i */

   if ( ( argc - optind ) < ONE )
   {
      PKIPATH_tpPrintUsage() ;
      return ( NEG ) ;
   } /* if argc */
   dn = strdup( argv[ optind ] ) ;

/* Call path determination/validation procedure */
   PKIPATH_tpPath( dn, fNeedKeyUsage, keyUsage ) ;
   return( ZERO ) ;
}

#endif /* _MAINPATH_ */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory read operation via LDAP
 *                with strong authentication (main program)
 *                Obs.: Use kinit (kerberos) to get a TGT before
 *                      running this program.
 *   Version:     1.0
 *   File:        mainread.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 24th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _MAINSRCH_
#define _MAINSRCH_

#include "search.h"

/* Prints correct usage of this program */
void  PKIREAD_tpPrintUsage( void )
{
   fprintf ( stderr,
             "\n------------- PKI LDAP READ OPERATION ----------------\n" ) ;
   fprintf ( stderr, "\nUsage: pkread [options] filter [attributes]\n" ) ;
   fprintf ( stderr, "options:\n" ) ;
   fprintf ( stderr, "\t-r\t\tdo not follow referrals\n" ) ;
   fprintf ( stderr, "\t-t\t\twrite binary values to files\n" ) ;
   fprintf ( stderr, "\t-k\t\tuse Kerberos authentication\n" ) ;
   fprintf ( stderr,
             "\t-b base\t\tsearch base dn (default is %s)\n" ,
             DEFAULT_BASE                                        ) ;
   fprintf ( stderr,
             "\t-h host\t\tLDAP server name (default is %s)\n" ,
             DEFAULT_HOST                                        ) ;
   fprintf ( stderr,
             "\t-p port\t\tLDAP port number (default is %d)\n" ,
             DEFAULT_PORT                                        ) ;
   fprintf ( stderr, "\t-u user\t\tuser's bind dn\n" ) ;
   fprintf ( stderr, "filter:\t\tLDAP search filter \"objectclass=...\"\n" ) ;
   fprintf ( stderr,
             "attributes:\tattributes to be retrieved (default is all)\n" ) ;
   fprintf ( stderr, "\t\tmax number of attributes = 6\n" ) ;
   fprintf ( stderr,
             "-------------------------------------------------------\n\a" ) ;
}
```

```
/* Main routine */

int main( int argc, char **argv )
{
   int                        i ;

   PKISRCH_tpSearchArgs       readArgs ;

/* Search arguments' default values */
   readArgs.user                = NULL ;
   readArgs.scope               =
   readArgs.originalscope       = LDAP_SCOPE_SUBTREE ;
   readArgs.base                =
   readArgs.parentbase          =
   readArgs.originalbase        = DEFAULT_BASE ;
   readArgs.host                = DEFAULT_HOST ;
   readArgs.port                = DEFAULT_PORT ;
   readArgs.auth                = NO_AUTH ;
   readArgs.fFollowReferral     = YES ;
   readArgs.fBinariesToFiles    = NO ;

/* Parse command line */
   while ( ( i = getopt( argc, argv, "rtkb:h:p:d:u:" ) ) != EOF )
   {
      switch ( i )
      {
         case 'r' :
         /* do not follow referrals */
            readArgs.fFollowReferral    = NO ;
            break ;

         case 't' :
         /* write binary values to files */
            readArgs.fBinariesToFiles   = YES ;
            break ;

         case 'k' :
         /* do kerberos authentication */
            readArgs.auth              = KRB_AUTH ;
            break ;

         case 'b' :
         /* search base */
            readArgs.originalbase      =
            readArgs.base              = strdup( optarg ) ;
            break ;

         case 'h' :
         /* LDAP host */
            readArgs.host              = strdup( optarg ) ;
            break ;

         case 'p' :
         /* LDAP port */
            readArgs.port              = atoi( optarg ) ;
            break ;

         case 'u' :
         /* user bind dn */
            readArgs.user              = strdup( optarg ) ;
            break ;

         default :
            if( argc > ONE )
            {
               PKIREAD_tpPrintUsage( ) ;
               return( NEG ) ;
```

```
            } /* if */
            break ;

      } /* switch */
   } /* while */

/* look for filter argument */
   if ( ( argc - optind ) < ONE )
   {
      PKIREAD_tpPrintUsage() ;
      return ( NEG ) ;
   } /* if argc */

   readArgs.filter         =
   readArgs.originalfilter = strdup( argv[ optind ] ) ;

/* look for attributes to be returned */
   optind++ ;
   if (argv[ optind ] == NULL )
   {
      readArgs.attrs       = NULL ;
   } else {
      readArgs.attrs       = &argv[ optind ] ;
   } /* if argv */

#ifdef _DEBUG_
   printf( "\nBeginning initial search...\n" ) ;
#endif

/* Perform initial search */
   PKIREAD_fFirstCall = YES ;
   PKISRCH_tpSearch( &readArgs ) ;

   return( ZERO );
}

#endif /* _MAINSRCH_ */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search operation via LDAP
 *                with strong authentication (main program)
 *                Obs.: Use kinit (kerberos) to get a TGT before
 *                      running this program.
 *   Version:     1.0
 *   File:        mainsrch.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 20th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _MAINSRCH_
#define _MAINSRCH_
#include "search.h"

/* Prints correct usage of this program */
void  PKISRCH_tpPrintUsage( void )
{
   fprintf ( stderr,
             "\n---------------- LDAP DISTRIBUTED SEARCH ----------------\n" )
;
   fprintf ( stderr, "\nUsage: pksearch [options] filter [attributes]\n" ) ;
   fprintf ( stderr, "options:\n" ) ;
   fprintf ( stderr, "\t-r\t\tdo not follow referrals\n" ) ;
   fprintf ( stderr, "\t-k\t\tuse Kerberos authentication\n" ) ;
   fprintf ( stderr,
             "\t-b base\t\tsearch base dn (default is \"o=PUC-Rio, c=BR\"\n" )
;
   fprintf ( stderr,
             "\t-s scope\tsearch scope (base, one or sub; default is sub)\n" )
;
   fprintf ( stderr,
             "\t-h host\t\tLDAP server name (default is localhost)\n" ) ;
   fprintf ( stderr, "\t-p port\t\tLDAP port number (default is 12345)\n" ) ;
   fprintf ( stderr, "\t-u user\t\tuser's bind dn\n" ) ;
   fprintf ( stderr, "filter:\t\tLDAP search filter\n" ) ;
   fprintf ( stderr,
             "attributes:\tattributes to be retrieved (default is all)\n" ) ;
   fprintf ( stderr, "\t\tmax number of attributes = 6\n" ) ;
   fprintf ( stderr,
             "--------------------------------------------------------\n\a" )
;
}
```

```
/* Main routine */

int main( int argc, char **argv )
{
   int          i ;

   PKISRCH_tpSearchArgs  srchArgs ;

/* Search arguments' default values */
   srchArgs.user              = NULL ;
   srchArgs.scope             =
   srchArgs.originalscope     = LDAP_SCOPE_SUBTREE ;
   srchArgs.base              =
   srchArgs.parentbase        =
   srchArgs.originalbase      = DEFAULT_BASE ;
   srchArgs.host              = DEFAULT_HOST ;
   srchArgs.port              = DEFAULT_PORT ;
   srchArgs.auth              = NO_AUTH ;
   srchArgs.fFollowReferral   = YES ;
   srchArgs.fBinariesToFiles  = NO ;

/* Parse command line */
   while ( ( i = getopt( argc, argv, "rtkb:s:h:p:d:u:" ) ) != EOF )
   {
      switch ( i )
      {
         case 'r' :
         /* do not follow referrals */
            srchArgs.fFollowReferral   = NO ;
            break ;

         case 't' :
         /* write binary values to files */
            srchArgs.fBinariesToFiles  = YES ;
            break ;

         case 'k' :
         /* do kerberos authentication */
            srchArgs.auth             = KRB_AUTH ;
            break ;

         case 'b' :
         /* search base */
            srchArgs.originalbase     =
            srchArgs.base             = strdup( optarg ) ;
            break ;

         case 's' :
         /* search scope */
            if ( strncasecmp( optarg, "base", 4 ) == ZERO )
            {
               srchArgs.originalscope =
               srchArgs.scope         = LDAP_SCOPE_BASE ;

            } else if ( strncasecmp( optarg, "one" , 3 ) == ZERO ) {

               srchArgs.scope         = LDAP_SCOPE_ONELEVEL ;

            } else if ( strncasecmp( optarg, "sub" , 3 ) == ZERO ) {

               srchArgs.scope         = LDAP_SCOPE_SUBTREE ;

            } else {
               fprintf( stderr ,
                        "--> Possible values of scope: base, one or sub\n" ) ;
               PKISRCH_tpPrintUsage( ) ;
               return( NEG ) ;
```

```
            } /* if */
            break ;

         case 'h' :
         /* LDAP host */
            srchArgs.host             = strdup( optarg ) ;
            break ;

         case 'p' :
         /* LDAP port */
            srchArgs.port             = atoi( optarg ) ;
            break ;

         case 'u' :
         /* user bind dn */
            srchArgs.user             = strdup( optarg ) ;
            break ;

         default :
            if( argc > ONE )
            {
               PKISRCH_tpPrintUsage( ) ;
               return( NEG ) ;
            } /* if */
            break ;

      } /* switch */
   } /* while */

/* look for filter argument */
   if ( ( argc - optind ) < ONE )
   {
      PKISRCH_tpPrintUsage( ) ;
      return( NEG ) ;
   } /* if */
/* search filter */
   srchArgs.originalfilter =
   srchArgs.filter         = strdup( argv[ optind ] ) ;

/* look for attributes to be returned */
   optind++ ;
   if (argv[ optind ] == NULL )
   {
      srchArgs.attrs       = NULL ;
   } else {
      srchArgs.attrs       = &argv[ optind ] ;
   } /* if */

#ifdef _DEBUG_
   printf( "\nBeginning initial search...\n" ) ;
#endif

/* Perform initial search */

   PKISRCH_tpSearch( &srchArgs ) ;

   return( ZERO );
}

#endif /* _MAINSRCH_ */
```

Módulos principais

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Certificate path processing (path determination
 *                and validation - PATH)
 *   Version:     1.0
 *   File:        path.cc
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 25th 1998
 *
 *      Revision 1.1 - Feb 24th 1999 17:48:35(GMT-3) arsilva
 *         Modified code to use Oscar's xcert ASN.1 structure
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _PATH_CC_
#define _PATH_CC_

#include "path.hh"

/* Path determination/validation routine */
int PKIPATH_tpPath( char*            dn ,
                    PKISRCH_tpfFlag  fNeedKeyUsage ,
                    KeyUsageType     keyUsage        )
{
   PKISRCH_tpfFlag       fFirstEntry ,
                         fMyRootFound ,
                         fGoFindXCerts ;

/* Argument structure used by PKI search & read operations */
   PKISRCH_tpSearchArgs* readArgs = new PKISRCH_tpSearchArgs() ;

/* Stores certificate path to be validated*/
   list<Certificate>     certs ;

   DistinguishedName     subject_dn ;
   DistinguishedName     issuer_dn ;
   DistinguishedName     myroot_dn ;

/* Initial settings */
   fFirstEntry          = YES ;
   fMyRootFound         =
   fGoFindXCerts        = NO ;
   myroot_dn            = DistinguishedName( MOST_TRUSTED ) ;
   subject_dn           = DistinguishedName( dn ) ;
   issuer_dn            = DistinguishedName("") ;
```

```
/* Try to find certificate path */
   while ( fMyRootFound == NO )
   {
      int                    i ;
      PKISRCH_tpfFlag        fPKIValueFound ;

   /* Initial read arguments */
      readArgs->user             = NULL ;
      readArgs->filter           = DEFAULT_FILTER ;
      readArgs->parentbase       = DEFAULT_BASE ;

      for ( i = ZERO ; i < MAX_PKIVALUES ; i++ )
      {
         pkivals[ i ].len  = ZERO ;
         pkivals[ i ].type = NONE ;
      } /* for i */

   /* Choose wich dn to look for within distributed repository */
      if ( fFirstEntry == YES )
      {
         readArgs->base          = dn ;
      } else {
         readArgs->base          = (char*) issuer_dn.toString().c_str() ;
      } /* if first */


      readArgs->host              = DEFAULT_HOST ;
      readArgs->port              = DEFAULT_PORT ;
      readArgs->auth              = NO_AUTH ;
      readArgs->fFollowReferral  =
      readArgs->fBinariesToFiles = YES ;

      #ifdef _PATH_DEBUG_
      cout << "Filter : " << readArgs->filter << endl ;
      cout << "Base   : " << readArgs->base   << endl ;
      cout << "Host   : " << readArgs->host   << endl ;
      cout << "Beginning read operation"      << endl ;
      #endif /* _PATH_DEBUG_ */

   /* Perform PKI (LDAP) read operation */
      PKIREAD_fFirstCall = YES ;
      if( PKISRCH_tpSearch( (PKISRCH_tpSearchArgs*) readArgs ) == NEG )
      {
         cerr << "Read operation failed: aborting" << endl ;
         return( NEG ) ;
      } /* if read ok */

      #ifdef _PATH_DEBUG_
      cout << "Read operation finished" << endl ;
      #endif /* _PATH_DEBUG_ */

      list<Certificate>    ucerts ;  /* user certificates list  */
      list<Certificate>    cacerts ; /* ca certificates list    */
      list<Certificate>    xcerts ;  /* cross certificates list */
   /* No CRLs yet :-( */

      i              = ZERO ;
      fPKIValueFound = NO ;

   /* Read all pki data files from last read operation */
      while ( ( pkivals[ i ].len != ZERO ) && ( fMyRootFound == NO ) )
      {

         #ifdef _PATH_DEBUG_
        cout << "PKI value found:"
              << " type= "   << pkivals[ i ].type
              << ", length=" << pkivals[ i ].len  << endl ;
```

```
   #endif /* _PATH_DEBUG_ */

   bytes                    data ;
   char                     filename[ MAX_FILENAMSIZE ] ;
   int                      filenum ,
                            len ,
                            j ;
   PKIREAD_tpPKIDataType    pkitype ;

   filenum  = i ;
   j        = ZERO ;
   pkitype  = pkivals[ i ].type ;

   len      = strlen( PKI_TMPFILENAME ) ;
   strncpy( filename, PKI_TMPFILENAME, len ) ;
   filename[ len ] = BYTE_ZERO ;

/* writes file number & type to filename template;    */
/*          converts number & type to strings         */
   while ( filenum > ZERO )
   {
      filename[ len - j - ONE ] = ( filenum % 10 ) + '0' ;
      filenum  = filenum / 10 ;
      j++ ;
   } /* while filenum */
   filename[ 5 ] = pkitype + '0' ;

/* Read tmp file 'filename' */
   if ( !data.readFile( filename ) )
   {
      cerr << "Could not open tmp file #"
           << i << ": " << filename << "; aborting" << endl ;
      return( NEG ) ;
   } /* if !data */

/* Delete tmp file 'filename' */
   if ( remove( filename ) != ZERO )
   {
     cerr << "Could not delete tmp file #"
          << i << ": " << filename << "; aborting" << endl ;
     return( NEG ) ;
   } /* remove */
/* DER decode data read from 'filename' */
   Certificate       cert ;
   CertificatePair   xcert ;

/* Store PKI data value in its respective list */
   switch ( pkivals[ i ].type )
   {
   /* it's a user certificate */
      case UCERT:
         cert.decode( data ) ;
         subject_dn = cert.getSubject() ;
         issuer_dn  = cert.getIssuer() ;
         ucerts.push_back(cert) ;
         fPKIValueFound = YES ;
         break ;

   /* it's a CA certificate */
      case CACERT:
         cert.decode( data ) ;
         subject_dn = cert.getSubject() ;
         issuer_dn  = cert.getIssuer() ;
      /* Root CA found: check if it is the most-trusted */
         fPKIValueFound = YES ;
         if ( issuer_dn == myroot_dn)
         {
            fMyRootFound = YES ;
```

```
            } else {
               if ( issuer_dn == subject_dn )
               {
                  fGoFindXCerts = YES ;
                  break ;
               } /* if self-signed certificate is not my root CA */
            } /* if issuer == subject */
            cacerts.push_back( cert ) ;
            break ;

      /* it's a cross certificate */
         case XCERTP:
            xcert.decode(data) ;
       /* if my root not found, put forward certificate */
         /*           in xcerts list; else, do nothing    */
            if ( fMyRootFound == NO )
            {
            /* only forward certificates are put in list */
               if ( xcert.hasForward() == NO )
               {
                  #ifdef _PATH_DEBUG_
                  cout << "No forward certificate found" << endl ;
                  #endif /* _PATH_DEBUG_ */
               } else {
                  xcerts.push_back( xcert.getForward() ) ;
               } /* xcert.hasForward? */
            } /* if fMyRootFound */
            fPKIValueFound = YES ;
            break ;

      /* it's a auth revoc list */
         case ALIST:

         /* Not addressed yet */
            break ;

      /* it's a cert revoc list */
         case CLIST:

         /* Not addressed yet */
            break ;

         default:
         /* Not a pki data value: do nothing */
            #ifdef _PATH_DEBUG_
            cout << "Non PKI related value found" << endl ;
            #endif /* _PATH_DEBUG_ */
            break ;
      } /* switch pkivals */

      #ifdef _PATH_DEBUG_
      cout << "Issuer : " << issuer_dn.toString( )  << endl ;
      cout << "Subject: " << subject_dn.toString( ) << endl ;
      #endif /* _PATH_DEBUG_ */

      i++ ;
   } /* while pkivals */

/* If no PKI value found at this point, then abort */
   if ( fPKIValueFound == NO )
   {
      cerr << "No PKI value found: aborting" << endl ;
      return( NEG ) ;
   } /* if list empty */

/* Certificate to compose path will be chosen from this list */
   list<Certificate>      tmpcerts ;
```

```
/* Decide from wich list one certificate to compose path will be chosen */
   if ( fFirstEntry == YES )
   {
      if ( ucerts.begin() != ucerts.end() )
      {
         tmpcerts     = ucerts ;
      } else {
         tmpcerts     = cacerts ;
      } /* if ucerts */
   } else {
      if ( fGoFindXCerts == YES )
      {
         tmpcerts      = xcerts ;
         fNeedKeyUsage = NO ;
      } else {
         tmpcerts      = cacerts ;
         keyUsage      = KEYUSAGE_KEYCERTSIGN ;
      } /* if fGoFindXCerts */
   } /* if fFirstEntry */

   list<Certificate>::iterator   k ;
   KeyUsage                      usage ;

   for ( k = tmpcerts.begin() ; k != tmpcerts.end() ; k++ )
   {

   /* Verify keyUsage extension to choose wich certificate to use;     */
   /*          if no keyUsage ext found, choose the first cert in list */
   /*          (so far, only one bit can be checked at a time)         */
      if ( fNeedKeyUsage == YES )
      {
         if ( (*k).hasKeyUsage() )
         {
            usage = (*k).getKeyUsage() ;
            if ( usage.hasKeyUsage( keyUsage ) )
            {
               #ifdef _PATH_DEBUG_
               cout << "Key usage verification ok"     << endl ;
               #endif /* _PATH_DEBUG_ */

               certs.push_front( (*k) ) ;
               break ;
            } else {
               cerr << "Key usage verification failed" << endl ;
            /* If last certificate and no valid keyUsage found, */
            /*          then abort                              */
               k++ ;
               if ( k == tmpcerts.end() )
               {
                  cerr << "No certificate with valid keyUsage found: "
                       << "aborting" << endl ;
                  return( NEG ) ;
               } /* if k */
               k-- ;
            }/* if usage.hasKeyUsage */
         } else {
            cerr << "Certificate with no key usage extension" << endl ;
         /* If last certificate and no valid keyUsage found, */
         /*          then abort                              */
            k++ ;
            if ( k == tmpcerts.end() )
            {
               cerr << "No certificate with valid keyUsage found: "
                    << "aborting" << endl ;
               return( NEG ) ;
            } /* if k */
            k-- ;
         } /* if (*k).hasKeyUsage */
```

```
         } else {
         /* Look for a cross certificate issued by my root CA */
            if ( fGoFindXCerts == YES )
            {
               if ( (*k).getIssuer() == myroot_dn )
               {
               /* Put cross certificate in certs list, if its */
               /*           issuer is my root CA              */
                  certs.push_front( (*k) ) ;
                  fGoFindXCerts = NO ;
                  fMyRootFound  = YES ;
                  break ;
               } /* if (*k) */
            } else {
            /* Put first certificate in certs list when */
            /*           no keyUsage extension needed   */

               #ifdef _PATH_DEBUG_
               cout << "No keyUsage extension needed; "
                    << "will use first certificate in list" << endl ;
               #endif /* _PATH_DEBUG_ */

               certs.push_front( (*k) ) ;
               break ;
            } /* if fGoFindXCerts */
         } /* if fNeedKeyUsage */
      } /* for k */

   /* If found a different root CA cert and no xcert issued by */
   /*           my root CA, then abort                         */
      if ( fGoFindXCerts == YES )
      {
         cerr << "No cross certificate found, issued by my root CA: "
              << "aborting" << endl ;
         return( NEG ) ;
      } /* if fGoFindXCerts */

      fFirstEntry = NO ;
   } /* while not found my root CA */

/* Put my root CA certificate in list to validate path */
   bytes                        tmpdata ;
   char*                        rootFilename ;
   Certificate                  myroot ;
   rootFilename = ROOT_FILENAME ;
   if ( !tmpdata.readFile( rootFilename ) )
   {
      cerr << "Could not open local root CA certificate: aborting" << endl ;
      return( NEG ) ;
   }
   myroot.decode( tmpdata ) ;

   certs.push_front( myroot ) ;

/* If there is a certification path, then validate it */
   if ( certs.begin() != ZERO )
   {
      list<Certificate>::iterator i ;
      int                         j ;

      j = ONE ;

      cout << endl << "==> Path determination complete; "
           << "the following path will be validated:" << endl ;
      for ( i = certs.begin() ; i != certs.end() ; i++ )
      {
         cout << j << ": " << (*i).getSubject().toString() << endl ;
         cout << "   issued by " << (*i).getIssuer().toString() << endl ;
```

```
         j++ ;
      } /* for i */

      CertificatePath    path(certs) ;

      if ( path.verify() )
      {
         cout << endl << "==> Valid path!!!" << endl ;
      } else {
         cerr << endl << "==> Invalid path!!! Problem at certificate from: "
              << path.getBadCertificate().getSubject().toString()  << endl
              << "==> Reason: " << path.getReason()                << endl ;
      } /* if path.verify */
   } /* if certs.begin */

   return( ZERO ) ;
}

#endif /* _PATH_CC_ */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search operation via LDAP
 *                with strong authentication (PKI repository search -
 *                SRCH)
 *   Version:     1.0
 *   File:        search.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 23th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*
 *   search.c
 *
 *   november 23th 1998 - version 1.61
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *               Alexandre Reis e Silva
 *
 *   Distributed directory search via LDAP with strong authentication.
 *   PKI repository search or read routine
 *
 *   Copyright (c) 1998 DI/PUC-Rio
 *   All rights reserved.
 */

#ifndef _SEARCH_
#define _SEARCH_

#include "search.h"

/* LDAP search routine */

int PKISRCH_tpSearch( PKISRCH_tpSearchArgs* args )
{

/* LDAP related variables */
   LDAP*           ld ;
   LDAPMessage*    entry ;
   LDAPMessage*    result ;
   struct timeval  timeout;

/* Other variables */
```

```
   int             i ,
                   baselen ,
                   hostlen ;
   char*           referralString ;
   char            temphost[ MAX_STRSIZE ] ;
   char            tempbase[ MAX_STRSIZE ] ;
   PKISRCH_tpfFlag fFoundReferral ,
                   fCanGoOn ,
                   fEntryFound ,
                   fNotSameBaseAgain ;

   #ifdef READ_ONLY
   int             len ;
   #endif /* READ_ONLY */

   fFoundReferral       =
   fCanGoOn             =
   fEntryFound          =
   fNotSameBaseAgain    = NO ;

/* Only for read operations */

   #ifdef READ_ONLY

   if ( PKIREAD_fFirstCall == YES )
   {
   /* Reset visited bases list */
      PKISRCH_VisitedBases[ ZERO ] = BYTE_ZERO ;

      args->originalbase  = args->base ;

   /* PKIX read operation's scope is always "base" */
      args->scope         =
      args->originalscope = LDAP_SCOPE_BASE ;

   /* PKIX read operation's filter is always "objectclass=..." */

      #ifdef _DEBUG_
      printf( "==> Filter : %s\n", args->filter ) ;
      printf( "==> Base   : %s\n", args->base ) ;
      printf( "==> Host   : %s\n", args->host ) ;
      #endif /* _DEBUG_ */

      len = strlen( args->filter ) ;

      #ifdef _DEBUG_
      printf( "==> Filter Length = %d\n", len ) ;
      #endif /* _DEBUG_ */

   /* If no no filter given, use default */
      if ( len == ZERO )
      {
         args->filter         =
         args->originalfilter = DEFAULT_FILTER ;
      } else {
      /* If filter given, verify its syntax */
         if (    (    strncmp( args->filter, "objectclass=", 12 )
                   == BYTE_ZERO )
              || (    strncmp( args->filter, "objectClass=", 12 )
                   == BYTE_ZERO ) )
         {
            args->originalfilter = args->filter ;
         } else {
            fprintf( stderr ,
                     "pkiread: %s: invalid search filter\n" ,
                     args->filter ) ;
            fprintf( stderr ,
                     "pkiread: only \"objectclass=*\" is allowed\n" ) ;
```

```
         return ( NEG ) ;
      } /* if strncmp */
   } /* if len */
   PKIREAD_fFirstCall = NO ;
} /* if PKIREAD_fFirstCall */

#else

/* Only read operation is allowed to write PKI binary values to files */
   args->fBinariesToFiles = NO ;

#endif /* READ_ONLY */

#ifdef _DEBUG_
printf( "\ntpSearch: Beginning to search... (args->fFollowReferral = %d)\n"
,
         args->fFollowReferral ) ;
#endif

/* Verify if base matches reference base to override LDAP server bug
          when search base does not match DIT base                */
   if ( PKISRCH_tpBaseMatch( args->base, args->parentbase ) < ZERO )
   {
      #ifdef _DEBUG_
      fprintf( stderr, "--> Search base (%s) does not match\n", args->base ) ;
      fprintf( stderr, "    parent base (%s)\n", args->parentbase ) ;
      #endif /* _DEBUG_ */

      if ( args->fFollowReferral == YES )
      {
         fprintf( stderr, "--> Base mismatch ; will try to find referrals\n" )
;
         args->base    = args->parentbase ;
         args->scope   = LDAP_SCOPE_SUBTREE ;
         args->filter  = REFERRAL_FILTER ;
      } /* if args->follow... */
   } /* if PKISRCH_tpBase... */

/* Search for all entries using given filter */
   while ( fCanGoOn == NO )
   {

      #ifdef _DEBUG_
      printf( "\ntpSearch: Getting LDAP handle ... host=%s, port=%d\n" ,
              args->host ,
              args->port ) ;
      #endif

   /* Store search base string in tempbase to override an LDAP bug
             in function ldap_open, that makes args->base disappear;
             tempbase is used as a ldap_search_st parameter          */
      baselen = strlen( args->base ) ;
      for( i = ZERO ; i < baselen ; i++ )
      {
         tempbase[ i ] = args->base[ i ] ;
      }
      tempbase[ i ] = BYTE_ZERO ;

      #ifdef _DEBUG_
      printf( "tpSearch: initial args->base = %s\n", args->base ) ;
      printf( "tpSearch: initial tempbase   = %s\n", tempbase   ) ;
      #endif

   /* Store search host string in temphost to override a LDAP bug
             in function ldap_open, that makes args->host disappear;
             temphost is used as a ldap_open parameter               */
      hostlen = strlen( args->host ) ;
      for( i = ZERO ; i < hostlen ; i++ )
```

```
      {
         temphost[ i ] = args->host[ i ] ;
      }
      temphost[ i ] = BYTE_ZERO ;

      #ifdef _DEBUG_
      printf( "tpSearch: initial args->host = %s\n", args->host ) ;
      printf( "tpSearch: initial temphost   = %s\n", temphost   ) ;
      #endif

   /* Get a handle to an LDAP connection */
      ld = ldap_open( temphost, args->port ) ;
      if ( ld  == NULL )
      {
         perror( "ldap_open" ) ;
         return( NEG ) ;
      }

      #ifdef _DEBUG_
      printf( "tpSearch: after ldap_open args->base = %s\n", args->base ) ;
      printf( "tpSearch: after ldap_open tempbase   = %s\n", tempbase   ) ;
      #endif

   /* Limits below are not used for read operation */
      ld->ld_sizelimit = SVR_SIZE_LIMIT ; /* max # of LDAP entries to return
*/
      ld->ld_timelimit = SVR_TIME_LIMIT ; /* deadline for server to respond
*/

      ld->ld_deref     = LDAP_DEREF_NEVER ; /* never deref alias (PKIX)
*/

   /* max time client waits for an answer */
      timeout.tv_sec  = CLI_TIME_LIMIT ;     /* seconds      */
      timeout.tv_usec = ZERO ;               /* micro-seconds */

      #ifdef _DEBUG_
      printf( "\ntpSearch: Initiating bind... (auth method=%d)\n" ,
               args->auth ) ;
      #endif

      if ( args->auth == KRB_AUTH )
      {

      /* Bind and authenticate to directory via Kerberos */
         if (   ldap_bind_s( ld, args->user, NULL, LDAP_AUTH_KRBV4 )
             != LDAP_SUCCESS )
         {
            ldap_perror( ld, "ldap_bind_s" ) ;
            return( NEG ) ;
         } /* if ldap_bind_s */

      } else {

      /* Bind anonymously to directory */
         if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS )
         {
            ldap_perror( ld, "ldap_simple_bind_s" ) ;
            return( NEG ) ;
         } /* if ldap_simple... */

      } /* if args->auth */

      #ifdef _DEBUG_
      printf( "\ntpSearch: Beginning search... base = %s, scope = %d\n" ,
              tempbase ,
              args->scope ) ;
      #endif
```

```
/* Address result codes after searching directory */
   switch ( ldap_search_st( ld ,
                            tempbase ,
                            args->scope ,
                            args->filter ,
                            args->attrs ,
                            ZERO ,
                            &timeout ,
                            &result ) )
   {

   /* Referrals are received via error message */
      case LDAP_PARTIAL_RESULTS :

         referralString    = strdup( ld->ld_error ) ;
         fFoundReferral    = YES ;
         args->filter      = args->originalfilter ;
         args->base        = args->originalbase ;
         args->scope       = args->originalscope ;

         #ifdef _DEBUG_
         printf( "\n--> %s\n", referralString ) ;
         #endif /* _DEBUG_ */

   /* No error code returned */
      case LDAP_SUCCESS :

         if ( ldap_count_entries( ld, result ) != ZERO )
         {
            printf( "\n--> LDAP search results:\n" ) ;
            printf( "\n    #    description\n\n" ) ;
            for ( entry  = ldap_first_entry( ld, result ) ;
                  entry != NULL ;
                  entry  = ldap_next_entry( ld, entry ) )
            {
               PKISRCH_tpOutputEntry( ld,  entry, args->fBinariesToFiles );

             } /* for entry */
             fEntryFound = YES ;
         } else {
            printf( "--> No entry found in server %s\n", temphost ) ;
         } /* if ldap_count... */
         fCanGoOn = YES ;
         break ;

   /* Although base does not belong to present DIT,
             it has a parent in there;
             ask DIT for referrals (search again)  */
      case LDAP_NO_SUCH_OBJECT :

         printf( "--> Search base not in this server\n" ) ;
         if (    ( args->fFollowReferral == YES )
              && ( fNotSameBaseAgain == NO      ) )
         {
            args->filter      = REFERRAL_FILTER ;
            args->base        = args->parentbase ;
            args->scope       = LDAP_SCOPE_SUBTREE ;
            fNotSameBaseAgain = YES ;
         } else {
            fCanGoOn = YES ;
         } /* if args->follow... */
         break ;
   /* Other result codes are errors */
      default :

         ldap_perror( ld, "ldap_search_st" ) ;
         return( NEG ) ;
```

```
      } /* switch */

   /* Free results storage structure */
      ldap_msgfree( result ) ;

   /* Unbind and free LDAP handle */
      ldap_unbind( ld ) ;

   } /* while */

/* Get URLs found in referral information */
   if ( fFoundReferral  && ( args->fFollowReferral == YES ) )
   {

      #ifdef _DEBUG_
      printf( "\ntpSearch: Beginning referral information gathering...\n" ) ;
      #endif

      if ( PKISRCH_tpGetUrl( referralString ) )
      {
      /* Parse referral URL strings found */
         i = ZERO ;
         while ( PKISRCH_urls[ i ][ ZERO ] != BYTE_ZERO )
         {
            if ( PKISRCH_tpParseReferral( PKISRCH_urls[ i ] ) )
            {
            /* new search parameters */
               args->parentbase  = PKISRCH_ludp->lud_dn ;
               args->host        = PKISRCH_ludp->lud_host ;
               args->port        = PKISRCH_ludp->lud_port ;

               #ifdef _DEBUG_
               printf( "\ntpSearch: Before: base= %s , parentbase= %s\n" ,
                       args->base ,
                       args->parentbase ) ;
               #endif

            /* Determine whether to propagate initial searchbase or not */
               if (   (    PKISRCH_tpBaseMatch( args->base, args->parentbase )
                        >= ZERO )
                   || (    PKISRCH_tpBaseMatch( args->parentbase, args->base )
                        >= ZERO ) )
               {
                  #ifdef _DEBUG_
                  printf( "\ntpSearch: args->base <- args->parentbase :
done\n" ) ;
                  #endif

                  args->base = args->originalbase /* former parentbase */ ;
               } /* if PKISRCH_tpBase... */

                  #ifdef _DEBUG_
                  printf( "\ntpSearch: After: base= %s , parentbase= %s\n" ,
                          args->base ,
                          args->parentbase ) ;
                  #endif

               if ( PKISRCH_tpVisited( args->parentbase ) == NO )
               {
               /* Search for entries where referrals indicate */
              printf( "--> Trying server %s...\n", args->host ) ;

               /* Search again - other base/server */
                  if ( PKISRCH_tpSearch( args ) == ZERO )
                  {
                     return( ZERO );
                  } /* if PKIREAD_tpPKIRead */
```

```
                #ifdef _DEBUG_
                } else {
                   printf( "--> %s: base already visited\n", args->parentbase )
;
                #endif
                } /* if PKISRCH_tpVisited */

                /* Free URL descriptor (from parseRef) */
/*                ldap_free_urldesc( PKISRCH_ludp ) ; */

             } /* if PKISRCH_tpParseReferral */
             i++ ;
          } /* while */
       } /* if PKISRCH_tpGetUrl */
   } /* if fFoundReferral*/

   if ( fEntryFound == YES )
   {
      return( ZERO ) ;
   } else {
      return( NEG ) ;
   } /* if fEntryFound */

}

#endif /* _PKIREAD_ */
```

Módulos de search.c

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search via LDAP with strong
 *                authentication (SRCH - referral addressing)
 *   Version:     1.0
 *   File:        referral.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 20th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _REFERRAL_
#define _REFERRAL_
#include "search.h"

/* Gets URLs from referral information */
int PKISRCH_tpGetUrl( char* referralString )
{
   int              i ,
                    j ,
                    first ,
                    last ,
                    len ;
   PKISRCH_tpfFlag  fQuote ;

   i         =
   j         =
   first     =
   last      = ZERO ;
   fQuote = NO ;

   len       = strlen( referralString ) ;

/* Initialize urls vector     */
   for ( i = ZERO ; i < MAX_ATTR ; i++ )
   {
      for ( j = ZERO ; j < MAX_REFSIZE ; j++ )
      {
         PKISRCH_urls[ i ][ j ] = BYTE_ZERO ;
      } /* for j */
   } /* for i */

   i = ZERO ;

/* Separate LDAP URL strings found in referral
```

```
            information  between quotes              */
   while ( last < len )
   {
      if ( referralString[ last ] == '"' )
      {
      /* Find quotes at position 'last' and sets fQuote */
         if ( fQuote == NO )
         {
            first  = ++last ;
            fQuote = YES ;
         } else {
         /* Copy string found to PKISRCH_urls vector  */
            for ( j = ZERO ; j < (last - first) ; j++ )
            {
               PKISRCH_urls[ i ][ j ] = referralString[ first + j ] ;
            } /* for j */
         /* Reset fQuote to try to find other URL strings */
            fQuote = NO ;
            i++ ;
         } /* if fQuote */
      } /* if referralString */
      last++ ;
   } /* while */

   return( i ) ;
}

/* Parses referrals found in previous search*/
int PKISRCH_tpParseReferral( char* url )
{
   int err;     /* Store parse error code */

   /* Verify if string is in URL format */
   if ( ! ldap_is_ldap_url( url ) )
   {
      fprintf( stderr, "String is not URL -> %s\n", url ) ;
      return( ZERO ) ;
   } /* if */

/* Verify if string is LDAP URL compliant and
            returns descriptor with parsing information */
   err = ldap_url_parse( url, &PKISRCH_ludp ) ;
   if ( err  != 0 )
   {
      fprintf( stderr, "String is not LDAP URL compliant -> %s\n", url ) ;
      fprintf( stderr, "Error code: %d\n", err ) ;
      return( ZERO ) ;
   } /* if */

   return( ONE );

}

#endif /* _REFERRAL_ */
```

```c
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search via LDAP with strong
 *                authentication (SRCH - base match verification:
 *                determines if a base is subordinated to another)
 *   Version:     1.0
 *   File:        bmatch.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 24th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _BASEMATCH_
#define _BASEMATCH_
#include "search.h"

/* Base match routine
 *
 * Return values: 1 - base matches refbase
 *                0 - base == parentBase
 *               -1 - base mismatch
 *
 * DN strings must follow syntax below:
 *    attr1=value1, attr2=value2, attr3=value3, ...
 *    - commas must be followed by one space character;
 *    - syntax is case insensitive.
 */
int PKISRCH_tpBaseMatch( char* newbase, char* parentbase )
{
   int     i ,
           newlen ,
           parentlen ;

   char    newb[ MAX_STRSIZE ] ,
           parent[ MAX_STRSIZE ] ;

   newlen    = strlen( newbase ) ;
   parentlen = strlen( parentbase ) ;

/* Pass newbase to new with lower case characters */
   for( i = ZERO ; i < newlen ; i++ )
   {
      newb[ i ] = tolower( newbase [ i ] );
   }
   newb[ i ] = BYTE_ZERO ;
```

```
/* Pass parentbase to parent with lower case characters */
   for( i = ZERO ; i < parentlen ; i++ )
   {
      parent[ i ] = tolower( parentbase [ i ] );
   }
   parent[ i ] = BYTE_ZERO ;

   #ifdef _DEBUG_
   printf( "tpBaseMatch: newbase=%s;\n", newb ) ;
   printf( "tpBaseMatch: parentbase=%s\n", parent ) ;
   #endif /* _DEBUG_ */

   if ( ( newlen == ZERO ) || ( parentlen == ZERO ) )
   {
      #ifdef _DEBUG_
      printf( "tpBaseMatch: parent base or new base with length = 0 !!!\n" ) ;
      #endif /* _DEBUG_ */
      return( NEG ) ;
   } else if ( strcmp( newb, parent ) == ZERO ) {
      #ifdef _DEBUG_
      printf( "tpBaseMatch: Base match ( newbase == parentbase )!!!\n" ) ;
      #endif /* _DEBUG_ */
      return( ZERO ) ;
   } else if ( strstr( newb, parent ) == NULL ) {
      #ifdef _DEBUG_
      printf( "tpBaseMatch: base mismatch!!!\n" ) ;
      #endif /* _DEBUG_ */
      return( NEG ) ;
   } else {
       #ifdef _DEBUG_
       printf( "tpBaseMatch: base match!!!\n" ) ;
       #endif /* _DEBUG_ */
       return( ONE ) ;
   } /* if */

   #ifdef _DEBUG_
   printf( "tpBaseMatch: ERROR!!!\n" ) ;
   #endif /* _DEBUG_ */
   return( NEG ) ;
}

#endif /* _BASEMATCH_ */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search via LDAP with strong
 *                authentication (SRCH - entry output)
 *   Version:     1.0
 *   File:        io.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 20th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _IOSTUFF_
#define _IOSTUFF_

#include "search.h"

/* Prints matching entries */
void PKISRCH_tpOutputEntry( LDAP*           ld ,
                            LDAPMessage*    entry ,
                            PKISRCH_tpfFlag fBinToFile )
{
   BerElement*                 ber ;
   struct berval**             bvals ;
   char*                       attrib ;
   char*                       dn ;
   int                         i ,
                               j ,
                               k ,
                               l ,
                               len ;
   PKISRCH_tpfFlag             fNotASCII ;

/* Variables related only to read operation */
   #ifdef READ_ONLY
   FILE*                       fp ;
   int                         filelen ,
                               filenum ;
   char                        filename[ MAX_FILENAMSIZE ] ;
   #endif /* READ_ONLY */

   i             =
   j             =
   k             =
   l             =
   len           = ZERO ;
```

```
   #ifdef READ_ONLY

/* Store filename template length */
   filelen       = strlen( PKI_TMPFILENAME ) ;

/* Reset PKI values data structure */
   for ( i = ZERO ; i < MAX_PKIVALUES ; i++ )
   {
      pkivals[ i ].type = NONE ;
      pkivals[ i ].len  = ZERO ;
   } /* for i */

   #endif /* READ_ONLY */

/* Get dn of each entry */
   dn = ldap_get_dn( ld, entry ) ;
      if ( dn != NULL )
   {
      count++ ;
      printf( "%6.3d\tdn: %s\n", count, dn ) ;
   } /* if dn */

/* Parse entries' attributes */
   for ( attrib = ldap_first_attribute( ld, entry, &ber ) ;
        attrib != NULL ;
         attrib = ldap_next_attribute( ld, entry, ber ) )
   {

   /* Get correspondig values of each attribute */
      bvals = ldap_get_values_len( ld, entry, attrib) ;
      if ( ( bvals ) != NULL )
      {
         for ( i = ZERO ; bvals[ i ] != NULL ; i++ )
         {
            fNotASCII = NO ;
         /* Verify if there are non-ascii characters */
            for ( j = ZERO ; j < bvals[ i ]->bv_len ; j++ )
            {
               if ( isascii( bvals[ i ]->bv_val[ j ] ) == BYTE_ZERO )
               {
                  fNotASCII = YES ;
                  break ;
               } /* if isascii */
            } /* for j */
            if ( fNotASCII == YES)
            {
               printf( "\t%s: NOT ASCII (%ld bytes) " ,
                       attrib ,
                       bvals[ i ]->bv_len                  ) ;

               #ifdef READ_ONLY

               if ( fBinToFile == YES )
               {
               /* find out if attribute is PKI related &                  */
               /*          put its respective value in "found" structure */
                  if ( strcmp( attrib, USER_CERT ) == ZERO )
                  {
                  /* attribute is userCertificate; copy its value */
                     pkivals[ k ].type = UCERT ;

                  } else  if ( strcmp( attrib, CA_CERT ) == ZERO ) {
                  /* attribute is cACertificate; copy its value */
                     pkivals[ k ].type = CACERT ;

                  } else  if ( strcmp( attrib, X_CERT ) == ZERO ) {
                  /* attribute is crossCertificatePair; copy its value */
```

```
                  pkivals[ k ].type = XCERTP ;

               } else  if ( strcmp( attrib, AUTH_LIST ) == ZERO ) {
               /* attribute is authorityRevocationList; copy its value */
                  pkivals[ k ].type = ALIST ;

               } else  if ( strcmp( attrib, CERT_LIST ) == ZERO ) {
               /* attribute is certificateRevocationList; copy its value */
                  pkivals[ k ].type = ALIST ;

               } else {
               /* attribute is not PKI related */
                  pkivals[ k ].type = NONE ;

               } /* if strcmp... */

            /* if PKI related, copy value to pkivals & write to a file */
               if ( pkivals[ k ].type != NONE )
               {
                  filenum = k ;
                  l = ZERO ;
                  strncpy( filename, PKI_TMPFILENAME, filelen ) ;
                  filename[ filelen ] = BYTE_ZERO ;

               /* Convert file number & type to strings & insert   */
               /*          them in filename,                       */
               /*          e.g.: type = UCERT, number= 12;         */
               /*                filename would be 'tmp_t1_n12'    */
                  while ( filenum > ZERO )
                  {
                     filename[ filelen - l - ONE ] = ( filenum % 10 ) + '0'
;
                     filenum  = filenum / 10 ;
                     l++ ;
                  } /* while filenum */
                  filename[ 5 ] = pkivals[ k ].type + '0' ;
                  fp = fopen( filename, "wb" ) ;
                  if ( fp  == NULL )
                  {
                     printf( "\n" ) ;
                     fprintf ( stderr ,
                        "\tI/O ERROR: could not open output file;\n" ) ;
                     fprintf ( stderr ,
                        "\t\tbinary data won't be written.\n" ) ;
                  } else {
                     pkivals[ k ].len = bvals[ i ]->bv_len ;
                     for( l = ZERO ; l < bvals[ i ]->bv_len ; l++ )
                     {
                        fputc( bvals[ i ]->bv_val[ l ], fp ) ;
                     } /* for l */
                     fclose( fp ) ;
                     printf( "- written to %s\n", filename ) ;
                     k++ ;
                  } /* if fp */
               } /* if pkivals */
            } /* if fBinToFile */

            #endif /* READ_ONLY */

         } else {
            printf ( "\t%s: %s\n", attrib, bvals[i]->bv_val ) ;
         } /* if fNotASCII */
      } /* for i */
      ber_bvecfree( bvals );
   } /* if bvals */
} /* for attrib */

printf( "\n" );
```

```
}

#endif /* _IOSTUFF_ */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search via LDAP with strong
 *                authentication (SRCH - visited remote bases
 *                verifying)
 *   Version:     1.0
 *   File:        visit.c
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 24th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _VISITED_
#define _VISITED_

#include "search.h"

/* Visited remote bases verification */

PKISRCH_tpfFlag PKISRCH_tpVisited( char* newbase )
{
   int             i ,
                   len ;

   char            newb[ MAX_STRSIZE ] ;

   #ifdef _DEBUG_
   printf( "tpVisited: Checking if base \"%s\" has already been visited\n" ,
           newbase ) ;
   #endif /* _DEBUG_ */

   len = strlen( newbase ) ;

   #ifdef _DEBUG_
   printf( "tpVisited: strlen done\n" ) ;
   #endif /* _DEBUG_ */

/* Pass newbase to newb with lower case characters */
   for( i = ZERO ; i < len ; i++ )
   {
      newb[ i ] = tolower( newbase [ i ] ) ;
   }
   newb[ i ] = BYTE_ZERO ;

   #ifdef _DEBUG_
```

```
   printf( "tpVisited: visited bases are... (just before checking)\n" ) ;
   i = ZERO ;
   while ( PKISRCH_VisitedBases[ i ] != BYTE_ZERO )
   {
      printf( "\t%2.2d) %s\n", i + 1, PKISRCH_VisitedBases[ i ] ) ;
      i++ ;
   } /* while */
   #endif /* _DEBUG_ */

   #ifdef _DEBUG_
   printf( "tpVisited: newb done: %s\n", newb ) ;
   #endif /* _DEBUG_ */

   i = ZERO ;

   #ifdef _DEBUG_
   printf( "tpVisited: comparing the following visited bases:\n" ) ;
   #endif /* _DEBUG_ */

   while ( PKISRCH_VisitedBases[ i ] != BYTE_ZERO )
   {
      #ifdef _DEBUG_
      printf( "\t%2.2d) %s\n", i, PKISRCH_VisitedBases[ i ] ) ;
      #endif /* _DEBUG_ */

      if ( strncmp( newb, PKISRCH_VisitedBases[ i ], len ) == ZERO )
      {
         #ifdef _DEBUG_
         printf( "tpVisited: base \"%s\" already visited\n", newb ) ;
         #endif /* _DEBUG_ */
         return( YES ) ;
      } /* if */
      i++ ;
   } /* while */

   #ifdef _DEBUG_
   printf( "tpVisited: base \"%s\" not visited\n", newb ) ;
   #endif /* _DEBUG_ */

   PKISRCH_VisitedBases[ i ] = strdup( newb ) ;

   #ifdef _DEBUG_
   printf( "tpVisited: visited bases are... (after checking)\n" ) ;
   i = ZERO ;
   while ( PKISRCH_VisitedBases[ i ] != BYTE_ZERO )
   {
      printf( "\t%2.2d) %s\n", i + 1, PKISRCH_VisitedBases[ i ] ) ;
      i++ ;
   } /* while */
   #endif /* _DEBUG_ */

   return( NO ) ;
}

#endif /* _VISITED_ */
```

Cabeçalhos

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Certificate path processing (path determination
 *                and validation - PATH)
 *   Version:     1.0
 *   File:        path.hh
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 25th 1998
 *
 *      Revision 1.1 - Feb 24th 1999 17:41:08(GMT-3) arsilva
 *         Included Oscar's xcert structure (CertificatePair.hh)
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _PATH_HH_
#define _PATH_HH_

/* System headers */
#include <list>
#include <string>
#include <stdio.h>
#include <iostream.h>

/* Oscar's headers */
#include <Certificate.hh>
#include <CertificatePath.hh>
#include <CertificatePair.hh>

/* LDAP distributed search/read headers */
#include "search.h"

#define _PATH_DEBUG_

/*
 * Constants
 */

/* Example dn to read its PKI values */
#define DEFAULT_NAME       "cn=andreia silva, ou=jur, o=puc-rio, c=br"

/* Example keyUsage to verify */
#define DEFAULT_USAGE      KEYUSAGE_DIGITALSIGNATURE

/* My most-trusted CA's distinguished name */
#define MOST_TRUSTED     "cn=rootCA, o=PUC-Rio, c=BR"
```

```
/* My most-trusted CA's certificate filename */
#define ROOT_FILENAME    "rootca.cert"

/*
 * Prototypes
 */

int   PKIPATH_tpPath( char*            dn ,
                      PKISRCH_tpfFlag  fNeedKeyUsage ,
                      KeyUsageType     keyUsage        ) ;

void  PKIPATH_tpPrintUsage( ) ;

#endif /* _PATH_HH_ */
```

```c
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *   Pontificia Universidade Catolica do Rio de Janeiro (PUC-Rio)
 *             Departamento de Informatica (DI)
 *
 *   Copyright (c) 1999 Alexandre R. Silva - DI/PUC-Rio
 *   All rights reserved.
 *
 *   This code is property of Alexandre R. Silva and it must
 *   be used strictly in accordance with the licence terms in
 *   LICENSE.TXT file.
 *
 *   If there is no LICENSE.TXT file in this distribution package,
 *   then you are not allowed to use this code at all and should
 *   contact the author, in order to get a license agreement.
 *
 *   This EXPERIMENTAL software is provided AS IS, without any kind
 *   of warranty, and must be used with extreme care. The author
 *   shall not be liable for any kind of damage, directly or
 *   indirectly caused by the use of this software.
 *
 *   Project:     Pequi (PKI)
 *   Description: Distributed directory search operation via LDAP
 *                with strong authentication (PKI repository search -
 *                SRCH)
 *   Version:     1.0
 *   File:        search.h
 *   Author:      Alexandre R. Silva (arsilva@inf.puc-rio.br)
 *
 *   History:
 *
 *      Created - Nov 20th 1998
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifndef _SEARCH_H_
#define _SEARCH_H_

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/*
 * System headers
 */

#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <stdlib.h>
#include <getopt.h>     /* To be used under RedHat */
#include <ctype.h>

/*
 * LDAP headers
 */

#include <lber.h>
#include <ldap.h>
#include <ldif.h>   /* LDAP SDK does not install this header automatically */

/* Enables program to output several messages */
/* #define _DEBUG_ */

/*
 * Constants
 */
```

```
#define DEFAULT_USER      "cn=Alexandre Silva, ou=INF, o=PUC-Rio, c=BR"

#ifndef DEFAULT_BASE
#define      DEFAULT_BASE "o=PUC-Rio, c=BR"     /* default search base (root)
*/
#endif /* DEFAULT_BASE */

#ifdef LDAPHOST
#define DEFAULT_HOST      LDAPHOST
#else
#define DEFAULT_HOST      "localhost"
#endif /* LDAPHOST */

#define DEFAULT_PORT      LDAP_PORT
#define DEFAULT_URL "\"ldap://olorum.inf.puc-rio.br:12345/o=PUC-Rio, c=BR\""
#define DEFAULT_BINDIR  "/home/arsilva/proj/certs/" /* binary files dir     */
#define REFERRAL_FILTER   "(objectclass=referral)"    /* for referral search
*/

#define BASE_OPTION "-b"   /* flag to indicate non-default search base */
#define      MAX_ATTR     6      /* max search attributes in command line
*/
#define MAX_NUMBASES    16      /* max number of visited search bases       */
#define MAX_RDNSIZE     64      /* max RDN string size                      */
#define MAX_STRSIZE 128    /* max generic string size               */
#define MAX_REFSIZE 512    /* max size of referral information         */
#define MAX_FILENAMSIZE 32      /* max filename string size                 */
#define SVR_SIZE_LIMIT    100    /* max entries to return
*/
#define SVR_TIME_LIMIT    60     /* max time for server to spend searching
*/
#define CLI_TIME_LIMIT    75     /* max time client waits for results
*/

#ifdef READ_ONLY

#define DEFAULT_FILTER  "objectclass=*"             /* default read filter  */
#define MAX_PKIVALUES   16                          /* max # of pki values  */

/* PKI related attributes */
#define USER_CERT       "usercertificate;binary"
#define CA_CERT         "cacertificate;binary"
#define X_CERT          "crosscertificatepair;binary"
#define AUTH_LIST       "authorityrevocationlist;binary"
#define CERT_LIST       "certificaterevocationlist;binary"

/* Filename template: tmp_t0_n00                      */
/*                         ^  ^^                      */
/*                        /     \                     */
/*           PKI data type       PKI data number      */
/*                                                    */
#define PKI_TMPFILENAME "tmp_t0_n00"

#endif /* READ_ONLY */

#define BYTE_ZERO   0x00

#define ONE             1
#define ZERO            0
#define NEG            -1

/*
 * Variables & structures
 */


/* For search & read operations */
```

```
/* Possible authentication methods */
   typedef enum
   {
      NO_AUTH ,
      KRB_AUTH
   } PKISRCH_tpAuthMethod ;

/* Possible values of any flag */
   typedef enum
   {
      NO ,
      YES
   } PKISRCH_tpfFlag ;

/* Search parameters passed to DD_Search */
   typedef struct
   {
      char*                  user ;
      char*                  originalfilter ;
      char*                  filter ;
      char*                  originalbase ;
      char*                  parentbase ;
      char*                  base ;
      int                    originalscope ;
      int                    scope ;
      char**                 attrs ;
      char*                  host ;
      int                    port ;
      PKISRCH_tpAuthMethod  auth ;
      PKISRCH_tpfFlag       fFollowReferral ;
      PKISRCH_tpfFlag       fBinariesToFiles ;
   } PKISRCH_tpSearchArgs ;

/* Stores URLs found in referral info */
   #if !defined( _SEARCH_ )
   extern
   #endif /* _SEARCH_ */
   char            PKISRCH_urls[MAX_ATTR][MAX_REFSIZE] ;

/* LDAP URL descriptor */
   #if !defined( _SEARCH_ )
   extern
   #endif /* _SEARCH_ */
   LDAPURLDesc*    PKISRCH_ludp ;

/* Stores visited LDAP bases */
   #if !defined( _VISITED_ )
   extern
   #endif /* _VISITED_ */
   char*           PKISRCH_VisitedBases[ MAX_NUMBASES ] ;

/* Stores number of entries found */
   #if defined( _IOSTUFF_ )
   static int          count ;
   #endif /* _IOSTUFF_  */


/* Only for read operations */

#ifdef READ_ONLY

/* Stores number of tpSearch calls */
   #if !defined( _SEARCH_ )
   extern
   #endif /* _SEARCH_ */
   PKISRCH_tpfFlag PKIREAD_fFirstCall ;
```

```
/* PKI data structure types */
   typedef enum
   {
      NONE ,
      UCERT ,
      CACERT ,
      XCERTP ,
      ALIST ,
      CLIST
   } PKIREAD_tpPKIDataType ;

/* Values of PKI related attributes found */
   typedef struct
   {
      PKIREAD_tpPKIDataType  type ;
      long int               len ;
   } PKIREAD_tpPKIValuesFound ;

/* Stores PKI values found */
   #if !defined( _SEARCH_ )
   extern
   #endif /* _SEARCH_ */
   PKIREAD_tpPKIValuesFound pkivals[ MAX_PKIVALUES ] ;

#endif /* READ_ONLY */

/*
 * Prototypes
 */

   int             PKISRCH_tpGetUrl( char *referralString ) ;
   int             PKISRCH_tpParseReferral( char *url ) ;
   int             PKISRCH_tpBaseMatch( char *newBase, char *parentBase ) ;
   int             PKISRCH_tpSearch( PKISRCH_tpSearchArgs* srchArgs ) ;

   PKISRCH_tpfFlag PKISRCH_tpVisited( char* base ) ;

   void            PKISRCH_tpToFile( char* bval ,
                                     int   blen ,
                                     char* filename ,
                                     char* dirname    ) ;

   void            PKISRCH_tpOutputEntry( LDAP*           ld ,
                                          LDAPMessage*    entry ,
                                          PKISRCH_tpfFlag binToFile ) ;


   #ifdef READ_ONLY
   void            PKIREAD_tpPrintUsage( void ) ;
   #else
   void            PKISRCH_tpPrintUsage( void ) ;
   #endif /* READ_ONLY */


#ifdef __cplusplus
}
#endif /* __cplusplus__ */

#endif /* _SEARCH_H_ */
```