UNIVERSITATEA “DUNĂREA DE JOS” GALAȚI
FACULTATEA DE ŞTIINȚA CALCULATOARELOR
SPECIALIZAREA: CALCULATOARE

Proiect de Diplomă

PROFESOR
Ş. L. Drd. Ing. Istrate Adrian

STUDENT
Presură George

- 2005 -

UNIVERSITATEA “DUNĂREA DE JOS” GALAȚI
FACULTATEA DE ŞTIINȚA CALCULATOARELOR
SPECIALIZAREA: CALCULATOARE
- Anul 5 -

SecureDoc

– Sistem electronic pentru –
– transferul securizat al documentelor –

PROFESOR
Ş. L. Drd. Ing. Istrate Adrian

STUDENT
Presură George

- 2005 -

Prefata

Necesitatea utilizarii unui sistem electronic de semnare/criptare documente este in continua crestere in paralel cu trecerea de la sistemul clasic de gestiune a documentelor la sistemul electronic.

Termenul „document” se refera la informatia stocata in format electronic si codificata sub forma de: email-uri, documente clasice, pdf-uri, prezentari, spreadsheet-uri, continut multimedia, etc.

Pentru a integra functiile de semnare / criptare / validare autenticitate semnatura digitala in aplicatiile curente de procesare documente este necesara un modul usor de utilizat, extensibil la cerinte viitoare fara a modifica insa conceptul de baza, si implementarea functiilor criptografice sa fie transparenta pentru utilizator.

Acesta este scopul aplicatiei SecureDoc.

Cuprins

Capitolul 1: Cerinte si specificatii

1.1 Enunt 4
1.2 Formularea cerintei 4

Capitolul 2: Analiza proiectului

2.1 Tehnologii folosite 6
2.2 Cercetari existente in domeniu 6
2.3 Viitorul solutiilor de securitate 7
2.4 Utilitare folosite in proiect 8

Capitolul 3: Implementare

3.1 Notiuni teoretice PKI 10
3.2 Functiile de baza implementate de aplicatie 15

Capitolul 4: Structura aplicatiei

4.1 Facilitatile aplicatiei 17
4.2 Structura fisierelor 19
4.3 Interfata utilizator 20
4.4 Implementarea functiilor aplicatiei 21
4.4.1 Operatia de semnare 21
4.4.2 Operatia de verificare semnatura 22
4.4.3 Operatia de criptare 22
4.4.4 Operatia de decriptare 23
4.4.5 Management repository 24
4.5 Comunicarea interfata-scripturi aplicatie 26

Capitolul 5: Manual de utilizare

5.1 Cerinte sistem 28
5.2 Instructiuni de instalare 29
5.3 Management repository 29
5.3.1 Management autoritati RootCA 32
5.3.2 Management autoritati SubCA 32
5.3.3 Management certificate End Entity ale tertilor utilizatori 33
5.3.4 Management certificate si chei private 34
5.3.5 Stergerea unui certificat/P12 din repository 35
5.3.6 Importul unui certificat/P12 in repository 37
5.3.7 Afisare detalii despre un certificat 37
5.3.8 Afisare detalii despre un fisier P12 40
5.4 Semnare documente 40
5.5 Criptare documente 42
5.6 Dezinstalare aplicatie 44

Capitolul 6: Bibliografie

Capitolul 7: Anexe

7.1 securedoc_sign 48
7.2 securedoc_crypt 51
7.3 securedoc_repository 53

Capitolul 1: Cerinte si specificatii

1.1 Enunt

Sistem software pentru lucrul cu documente semnate/criptate electronic.

1.2 Formularea cerintei

Sa se creeze un modul pentru suite de aplicatii OpenOffice care sa ofere suport pentru lucrul cu documente semnate si/sau criptate electronic.

Aplicatia trebuie sa ofere urmatoarele functionalitati:

- sa permita semnarea dpcumentelor electronice conform standardului SMIME
- sa permita criptarea documentelor electronice conform standardului SMIME
- sa permita managementului unui repository global (disponibil tuturor utilizatorilor sistemului)
- sa permita managementul unui repository local, specific fiecarui utilizator

Capitolul 2: Analiza proiectului

2.1 Tehnologii folosite

PKI (Public Key Infrastructure)

A fost folosita aceasta tehnologie pentru schimbul de documente securizate intre utilizatori datorita nivelului de securitate ridicat fata de metoda clasica de utilizare a unui sistem cu chei simetrice.

OpenOffice SDK

Reprezinta platforma pe care s-a dezvoltat aplicatia.

Versiunea SDK-ului utilizata este 1.1.4, si poate fi downloadat-a de pe site-ul www.openoffice.org.

Java

Limbajul de programare Java a fost ales pentru a dezvolta filtrul de fisiere OpenOffice. A fost ales acest limbaj in defavoarea C++ datorita simplitatii programarii, a documentarii functiilor OpenOffice SDK si a portabilitatii.

Linux

Pentru dezvoltarea si testarea aplicatiei s-a folosit sistemul de operare Linux. Interoperabilitatea OpenOffice – OpenSSL s-a facut prin intermediul unor scripturi bash. A fost necesara aceasta metoda deoarece OpenSSL nu pune la dispozitie un API Java .

2.2 Cercetari existente in domeniu

In urma unor cercetari aprofundate in domeniu (cu ajutorul prietenului Google) am gasit si nu am gasit aplicatii similare cu ceea ce s-a propus a fi acest proiect.

Incepand cu versiunea 2.0 a OpenOffice (care la momentul dezvoltarii acestui proiect era inca in stadiu de dezvoltare) s-a anuntat suport pentru semnare / criptare a documentelor direct din OpenOffice. Aceasta facilitate ar urma sa fie implementata in noul format al fisierelor OpenOffice (care are la baza standardul XML), insa functiile criptografice urmand a se efectua cu utilitare cu un nivel scazut de flexibilitate.

Astfel, chiar si pentru versiunea 2.0 a OpenOffice, acest utilitar va reprezenta o solutie alternativa pentru lucrul cu documente semnate / criptate (deci nu am lucrat chiar degeaba).

2.3 Viitorul solutiilor de securitate

Viitorul este stralucitor pentru astfel de aplicatii. Spun asta bazandu-ma pe doua puncte solide de sprijin:

- puterea tot mai ridicata de procesare a dispozitivelor electronice
- nivelul de siguranta tot mai ridicat cerut de utilizatori

Puterea ridicata de procesare a dispozitivelor electronice din zilele noastre (si cele care vor veni) permit efectuarea operatiilor criptografice intr-un interval de timp insesizabil de catre utilizator (cat ai clipi). Un alt factor care face facila utilizarea acestui standard este disponibilitatea API-urilor pentru toate limbajele de programarea utilizate la dezvoltarea de aplicatii pentru orice tip de dispozitiv electronic: calculatoare, telefoane mobile, PDA-uri, dispozitive de retea si telecomunicatii, bancomate, etc.

Nivelul de siguranta cerut de utilizatori este dealtfel cel mai important factor care a dus si duce in continuare greul promovarii tehnologiilor de criptare a datelor schimbate in interactiunile om-om, om-masina si masina-masina.

Interactiunile om-om reprezinta schimburile de date sub forma de documente in format electronic (email-uri, contracte, ordine de plata, informatii cu caracter confidential, etc.). Schimbul securizat de informatie in acest caz este determinat de caracterul confidential al datelor schimbate intre cei doi utilizatori, semnarea electronica fiind utilizata pentru a garanta autenticitatea documentului, iar criptarea pentru a preveni interventia hotului din mijloc.

Interactiunile om-masina sunt reprezentate de: sistemele de acces pe baza de cartela, bancomat-urile, efectuarea de plati in timp real cu dispozitive de tip smart-card (implementate fie sub forma de carti de credit, fie in telefoane mobile, laptop-uri, etc.), accesul la terminale de lucru cu smart-card-uri, paginile web securizate (https://), etc. Schimbul securizat de informatie in acest caz este determinat pe de o

parte de necesitatea de garantare a autenticitatii unei tranzactii (utilizandu-se semnarea acesteia de catre partile implicate) sau de se transmitere intr-un mod securizat de informatii confidentiale (numere de carti de credit) de la utilizator catre masina.

Interactiunile masina-masina consta in schimbul automat de informatii de configurare intre dispozitive hardware precum: echipamente de retea (firewall-uri, routere), terminale GPRS, etc. In aceste cazuri se folosesc in special operatiile de criptare date (schimbare tabele de routare, tabele de acces pentru firewall-uri, tranzactii intre bancomate si serverele bancii, etc.).

2.4 Utilitare folosite in proiect

Pentru implementarea operatiilor criptografice s-a utilizat aplicatia OpenSSL (versiunea 0.9.7g). Descarcarea acestei aplicatii se face de pe site-ul www.openssl.org.

Capitolul 3: Implementare

3.1 Notiuni teoretice PKI

In acest capitol voi descrie in ce consta tehnologia PKI, care sunt metodele de implementare si in ce consta avantajele aduse utilizatorilor (in special in interactiunile om-om, care reprezinta subiectul acestui proiect).

Mai intai, trebuie sa facem cunostinta cu trei persoane: Alice, Bob si Eve. Pe parcursul documentului, aceste nume vor face referire la urmatoarele persoane:

- Alice este persoana care doreste sa trimita un mesaj semnat / criptat
- Bob este persoana careia ii este destinat documentul trimis de Alice
- Eve joaca rolul de "man-in-the-middle" (iar de dragul lui Pruteanu vom traduce: omul din mijloc) care incearca fie sa altereze semnatura, fie sa decodifice datele pe care Alice le trimite lui Bob.

In prezent exista doua tehnici de criptare / transfer securizat a documentelor in format electronic: criptografia cu chei publice si cu chei simetrice. In randurile urmatoare voi evidentia avantajele si dezavantajele fiecarei tehnologii.

Criptarea cu chei simetrice

Pana la mijlocul anilor 1970 singurul mecanism cunoscut de catre specialisti pentru a transmite informatii intre doua puncte (calculatoare, sisteme de comunicatie, scrisori, etc.) era criptarea acestora utilizand o cheie cunoascuta de ambele parti implicate in procesul de comunicare. Operatia de criptare efectiva consta in transformarea datelor dupa un sistem (cipher) care utiliza un algoritm reversibil. Complexitatea „spargerii” mesajelor criptate consta de cele mai multe ori in determinarea algoritmului de criptare, apoi determinarea cheii utilizate pentru trasmiterea mesajului fiind o problema de timp, putand fi decodificat in cateva ore sau maxim zile.

Astfel, pentru o comunicatie sigura intre doua parti trebuie ca acestea sa cunoasca atat algoritmul de criptare cat si cheia secreta. Dezavantajele acestui sistem au inceput sa se faca simtite in momentul in care se doreste comunicarea intre mai multe parti, datorita dificultatii de scalare a acestui sistem. Deoarece fiecare persoana doreste ca mesajele pe care le schimba cu ceilalti sa nu poata fi decodificate de catre oricine altcineva decat destinatarul, atunci cel care trimite mesajul trebuie sa aiba cate o cheie secreta pentru fiecare utilizator cu care schimba mesaje criptate. Astfel, intr-un grup

de 10 persoane, fiecare utilizator trebuie sa gestioneze cate 9 chei secrete, iar numarul total de chei schimbate in grup este de 10 * 9.

O solutie pentru aceasta problema a fost introducerea unui server central, KDC (Key Distribution Center), care avea avantajul de a limita numarul de chei din sistem la N (numarul de utilizatori) si avea rolul de a face comunicarea intre doi utilizatori care nu facusera inainte schimb de chei. Acesta functiona astfel: cand un utilizator doreste sa trimita un mesaj catre un alt utilizator din sistem, acesta trimite mai intai o cerere la serverul KDC, criptata cu cheia lui secreta (pe care o cunoste doar el si serverul), apoi serverul creaza o cheie generata aleator pe care o trimite atat utilizatorului care a initiat cererea cat si utilizatorului caruia ii este destinat mesajul criptat. Dupa ce a primit mesajul de raspuns cu cheia generata de server, utilizatorul expeditor cripteaza mesajul cu aceasta cheie si il trimite catre destinatar.

Infrastructura cu chei publice

La mijlocul anilor 1970 doi cercetatori pe nume Whitfield Diffie si Martin Hellman au prezentat comunitatii studiile pe care le-au facut in domeniul algoritmilor de criptare asimetrici. Principalul avantaj al acestui sistem era ca se putea utiliza o pereche de chei, una pentru criptare si una pentru decriptare, dependente una de cealalta, dar aproape imposibil de calculat una din ele daca se cunostea cealalta. Astfel, una dintre chei se poate face publica si stocata in domeniul public iar cealalta va fi cheia privata, cunoscuta numai de catre posesor. Imposibilitatea practica a calcularii cheii private din cea publica este factorul principal care asigura securitatea acestui sistem. Astfel, pe masura ce puterea de calcul a sistemelor creste, trebuie ca si lungimea, in biti, a cheilor, trebuie sa creasca. In acest moment, cele mai folosite chei sunt cele cu lungimea de 128, 256, 512 si 1024 biti.

Tehnologia PKI ofera pe langa specificatiile algoritmilor de criptare, si o serie de servicii conexe, acestea fiind:

- criptare
- semnare
- asigurarea integritatii datelor
- faciliteaza schimbul de chei simetrice
- autoritate de ceritificare

Algoritmi utilizati in PKI

Operatiile criptografice utilizeaza o serie de algoritmi pentru operatiile de: criptare, generare hash-uri, generare chei, criptare simetrica, etc.

RSA

A fost inventat de Ron Rivest, Adi Shamir si Len Adleman in 1978, este unul dintre primii si cei mai flexibili algoritmi cu chei publice. Este utilizat pentru operatii de: criptare/decriptare, semnare/verificare semnatura, asigurarea integritatii datelor (prin semnare), schimb de chei. Securitatea acestui algoritm se bazeaza pe imposibilitatea tehnica de a calcula numere intregi factoriale de valori foarte mari. In acest moment s-a determinat ca o lungime a cheilor de 1024 biti este suficienta pentru viitorul apropiat si indelungat.

DSA

Digital Signature Algorithm este standardul NIST (National Institute of Standards and Technology, Statele Unite). Reprezinta o varianta a algoritmului ElGamal, proiectat pentru functii de semnare/verificare semnatura (si implicit asigurarea integritatii datelor). Securitatea acestui algoritm se bazeaza pe imposibilitatea tehnica de a calcula logaritmi intr-un interval finit. O lungime a cheilor optima pentru viitorul apropiat / indelungat este de 1024 biti.

DH

Acesta a fost propus de Whitfield Diffie si Martin Hellman si este utilizat exclusiv pentru operatiile de schimbare de chei. Fiecare parte utilizeaza cheia sa privata si cheia publica a corespondentului pentru a crea o cheie simetrica pe care nici un alt utilizator nu o poate calcula. Securitatea acestui algoritm se bazeaza pe imposibilitatea tehnica de a calcula logaritmi intr-un interval finit. O lungime a cheii de 1024 biti ar trebui sa fie suficienta pe termen mediu / indelungat.

SHA-1

The Secure Hash Algorithm SHA-1 este un algoritm de generare hash-uri pentru un mesaj. Lungimea in biti a hash-ului este de 160 biti. Hash-ul unui mesaj este folosit pentru operatia de semnare (care se efectueaza prin criptarea hash-ului cu cheia privata a utilizatorului, verificarea semnaturii facandu-se prin aplicarea cheii publice

extrasa din certificatul utilizatorului care a semnat mesajul, si comparandu-se apoi rezultatul cu hash-ul calculat dupa acelasi algoritm de catre destinatar.

Certificate digitale

Un certificat digital reprezinta o colectie de informatii codificate intr-un fisier in format PEM (codificat Base64) sau intr-un fisier DER (format binar). Avantajul formatului PEM este ca poate fi usor transmis pe internet utilizand protocoale de transmisie a datelor ce folosesc subsetul de caractere ASCII pe 7 biti.

In prezent sunt mai multe formate de certificate, cele mai raspandite fiind:

- certificate X509 Public-key
- certificate Simple Public Key Infrastructure (SPKI)
- certificate Pretty Good Privacy (PGP)
- certificate formate din atribute

Standardul cel mai raspandit si impus de companiile private din domeniul PKI este X509, si versiunea utilizata in prezent este versiunea 3.

Structura unui certificat digital in format X509 contine urmatoarele campuri:

- *Version*: versiunea formatului de certificat
- *Serial Number*: identificator unic al certificatului, atribuit de catre autoritatea de certificare (cea care a semnat acest certificat)
- *Signature*: contine identificatorul algoritmului utilizat pentru calcularea semnaturii certificatului
- *Issuer*: numele autoritatii care a semnat certificatul
- *Validity*: reprezinta intervalul de valabilitate al certificatului; este alcatuit din doua date, si anume: Not Valid Before si Not Valid After; aceste doua campuri contin data emiterii si expirarii certificatului, iar interpretarea lor de catre utilizatori este ambigua in cazul in care acestia nu au data sistemului corecta
- *Subject*: numele certificatului (in cazul certificatelor de servere acesta trebuie sa contina numele de domeniu sau adresa IP a serverului; in cazul certificatelor de email acesta va contine adresa de email a utilizatorului)
- *Subject Public Key Info*: contine cheia publica din perechea de chei a utilizatorului (cea care face pereche cu cheia privata a utilizatorului)
- *Issuer Unique Identifier*: cod de identificare unic al autoritatii de certificare (acest camp nu este de obicei utilizat)

- *Subject Unique Identifier*: cod de identificare unic al certificatului utilizatorului (acesta nu este recomandat a se utiliza, aceeasi functionalitate fiind oferita si de numarul serial al certificatului)

- *Extensions*: camp aditional cu extensii adaugat in versiunea 3 de certificate X509

Lista extensiilor pentru versiunea 3 a certificatelor X509 cuprinde:

- *Authority Key Identifier*: identificator unic al cheii utilizate pentru verificarea semnaturii digitale a certificatului (acesta camp este utilizat pentru a face distinctie intre cheile utilizate de *Issuer* pentru a semna acest certificat)

- *Subject Key Identifier*: identificator unic al cheii publice prezente in certificat (acest camp este utilizat pentru a face distinctie intre mai multe chei ale posesorului certificatului)

- *Key Usage*: reprezinta o masca de biti care specifica functionalitatile pentru care a fost creat acest certificat; functiile definite de acest camp sunt: digital signature, non-repudiation, key encipherment, data encipherment, key agreement, certificate signature, CRL signature, encipher only si decipher only

- *Extended Key Usage*: reprezinta o secventa de unul sau mai multi identificatori (OID) care identifica scopuri de utilizare ale cheii publice din certificat

- *CRL Distribution Point*: reprezinta o lista de URI (Uniform Resource Locator) cu adresele de unde pot fi descarcate listele cu certificatele revocate ale autoritatii care a creat acest certificat

- *Private Key Usage Period*: specifica intervalul de timp in care cheia privata pereche a cheii publice a certificatului poate fi utilizat; formatul acestui camp este asemanator cu cel al campului *Validity* al certificatului

- *Certificate Policies*: reprezinta o secventa de unul sau mai multi identificatori (OID) si calificatori asociati cu acestia care au scopul de a specifica aplicatiei care proceseaza certificatul date despre circumstantele in care certificatul respectiv poate fi utilizat

- *Policy Mappings*: contine o secventa de mapari (asocieri) dintre diferite policy-uri ale doua autoritati de certificare (acest camp este utilizat pentru a permite utilizarea pentru anumite servicii a unui certificat emis de o alta autoritate de certificare cu care s-a facut in prealabil cross-certificare)

- *Subject Alternative Name*: reprezinta un nume alternativ pentru certificat (cum ar fi adresa de email, adresa de domeniu sau IP, etc.)

- *Issuer Alternative Name*: reprezinta un nume alternativ asociat cu autoritatea care e eliberat certificatul (*Issuer*)

- *Subject Directory Attributes*: contine o secventa de atribute asociate cu posesorul certificatului

- *Basic Constraints*: specifica daca virgula certificatul este un certificat de autoritate superioara (adica nu este certificat de End Entity)

- *Path Length Constraint*: in cazul in care extensia *Basic Constraints* specifica un certificat de autoritate de certificare, atunci acest camp specifica numarul maxim de autoritati de certificare ce pot urma acestui certificat in lantul de certificate (altfel spus, restrictioneaza sub-autoritatile sale sa emita un anumit numar de sub-autoritati de certificare)

- *Name Constraints*: aceasta extensie este prezenta numai in certificatele autoritatilor (si nu prea am inteles la ce se refera ... erau niste cuvinte in engleza cam intortocheate ...)

- *Policy Constraints*: aceasta extensie este prezenta numai in certificatele autoritatilor de certificare (la fel ca mai sus)

3.2 Functiile de baza PKI implementate de aplicatie

Functiile de baza PKI implementate de aplicatie sunt:

a. semnare
b. verificare validitate / autenticitate semnatura
c. criptare
d. decriptare
e. management entitati (certificatele persoanelor cu care interactioneaza utilizatorul) si identitati (certificate si chei private ale utilizatorului curent).

Capitolul 4: Structura aplicatiei

4.1 Facilitatile aplicatiei

Aplicatia va oferi utilizatorilor urmatoarele functionalitati:

1) semnare document
2) listare si validare a persoanelor care au semnat documentul a si semnaturilor acestora
3) criptare document
4) decriptare document
5) managementul identitatilor utilizatorului si a entitatilor externe (operatii de instalare / dezinstalare certificate utilizator sau certificate entitati externe)
6) inspectare validitate document si vizualizarea istoricului operatiilor criptografice efectuate asupra sa

Semnarea documentelor

Aceata operatie se aplica pe un fisier deschis si salvat in OpenOffice si consta in crearea unui fisier cu acelasi nume si extensie ca fisierul original la care se adauga extensia ".p7s". Se obtine astfel un fisier care poate fi decodificat si modificat de orice persoana care intercepteaza acest fisier, insa destinatarul poate verifica daca acest fisier este intradevar semnat de catre persoana la care acesta astepta fisierul.

Verificarea validitatii / autenticitatii semnaturii

Procesul de verificare a semnaturii de catre Bob decurge astfel:

- salveaza fisierul primit si extrage certificatele cu care acesta a fost semnat
- verifica daca semnaturile aplicate de aceste certificate sunt valide (daca intradevar fisierul a fost semnat de catre acel certificat)
- verifica fiecare certificat pentru a determina daca are incredere in el sau nu

Ultima etapa daca nu se verifica poate duce la urmatorul scenariu: Eve (care vine de la Evil si ne duce cu gandul la Satana) sa intercepteze documentul si sa il modifice astfel incat Bob sa primeasca un document PKCS#7 semnat de catre Eve. Dar problema nu consta numai in acest lucru, ci poate apare in cazul in care Eve isi creeaza un certificat si o autoritate cu care semneaza acel certificat, iar acestea care sa aiba numele identice cu numele certificatului si autoritatii certificatului lui Alice. Astfel, pentru a preveni aceasta situatie, Bob trebuie sa verifice ca virgula certificatul

cu a carei cheie privata a fost semnat documentul primit sa fie semnat de o autoritate de incredere autoritate de incredere.

Criptarea documentelor

Aceasta operatie se aplica pe un fisier deschis si salvat in OpenOffice si consta in crearea unui fisier cu acelasi nume si extensie ca fisierul original la care se adauga extensia ".p7m". Se obtine astfel un fisier care poate fi decriptat doar de catre persoanele posesoare ale certificatelor din lista de recipienti ai documentului.

Operatie de criptare decurge astfel: Alice selecteaza o lista de entities (certificate de utilizatori externi) care vor fi adaugate in lista recipients a documentului criptat. Se observa astfel ca daca unul din certificatele din lista de recipients este prin diverse metode, un certificat "spion" inserat de Eve, atunci Eve va putea sa decodifice continutul acestui document. Deci trebuie ca Alice sa selecteze cu atentie lista de recipients, si deasemeni sa tina lista de entities intr-un loc securizat.

Decriptarea documentelor

Pentru a decripta un document, utilizatorul destinatie, Bob, va trebui sa aleaga o identitate (fisier PKCS#12) cu care sa decripteze documentul. Fisierul PKCS#12 contine cheia privata pereche a (unei) cheii publice cu care a fost criptat documentul.

Management repository

Permite gestionarea a patru locatii de stocare pentru diferite tipuri de date, si anume: Trusted Root CA, Trusted Intermediate CA, Other People si Personal.

Trusted Root CA reprezinta repository-ul cu certificatele autoritatilor de ROOT. Acestea sunt certificate autosemnate, disponibile pentru toti utilizatorii sistemului, si sunt stocate in directorul /etc/ssl/certs. Numele si structura acestor fisiere este specificata in documentatia OpenSSL.

Trusted Intermediate CA reprezinta repository-ul cu certificatele autoritatilor intermediare (SUBCA). Acestea sunt disponibile pentru toti utilizatorii sistemului, si sunt stocate in directorul /etc/ssl/certs. Numele si structura acetor fisiere este specificata in documentatia OpenSSL.

Other People reprezinta repository-ul cu certificatele persoanelor terte. Aceste certificate trebuie sa fie in format PEM si sunt folosite in special pentru criptare

(atunci cand se doreste trimiterea de mesaje criptate catre un utilizator). Sunt stocate in directorul ~/.securedoc/repository/entities al fiecarui utilizator. Accesul la acest director trebuie restrictionat pentru alte persoane.

Personal reprezinta repository-ul cu fisierele PKCS#12 care contin certificatele si cheile private (criptate) ale utilizatorului curent. Sunt stocate in directorul ~/.securedoc/repository/identities al fiecarui utilizator. Accesul la acest director trebuie restrictionat pentru alte persoane.

4.2 Structura fisierelor

Fisierele aplicatiei sunt organizate in doua grupuri: fisierele comune care se instaleaza in directorul OpenOffice, si fisierele cu repository-ul fiecarui utilizator.

Fisierele comune tuturor utilizatorilor sunt cele care se instaleaza in subdirectorul `./usr` din directorul aplicatiei OpenOffice.

Acestea contin:

- setarile necesare afisarii barei cu cele 4 butoane ale modulului SecureDoc in interfata OpenOffice, fisiere localizate in directorul

`/opt/ooo-1.1.4/user/config/soffice.cfg`

care contine fisierul userdeftoolbox1.xml cu specificatiile toolbar-ului cu butoanele modulului SecureDoc si subdirectorul `Bitmaps` care contine iconitele butoanelor.

- fisierele care contin specificatiile interfetelor grafice si codul OpenOffice Basic ce ofera functionalitatea interfetei grafice; acestea sunt localizate in directorul

`/opt/ooo-1.1.4/user/basic/Standard/`

si au numele: `securedoc_crypt.xba`, `securedoc_info.xba`, `securedoc_repository.xba` si `securedoc_sign.xba`.

Fisierele care formeaza repository-ul privat al utilizatorilor sunt stocate in dorectoarele home ale acestora.

Astfel, din kitul aplicatiei va fi copiat subdirectorul .securedoc in directoarele fiecaror utilizatori. Acesta are urmatoarea structura componenta:

- directorul `./repository/entities` contine repository-ul cu certificatele tertilor utilizatori; la instalare, continutul acestui director poate fi populat de catre administratori cu certificatele de End Entity din structura PKI a organizatiei respective

- directorul `./repository/identities` contine repository-ul cu fisierele P12 ale fiecarui utilizator; la instalare, continutul acestui director este gol pusca

- directorul ./tmp care va fi folosit de catre componentele aplicatiei pentru a stoca fisiere temporare necesare pentru procesarea operatiilor criptografice

- fisierele din directorul `~/.securedoc`, care contin scripturile care efectueaza operatiile criptografice.

4.3 Interfata utilizator

Implementarea interfetei utilizator s-a facut utilizand designer-ul de interfete al OpenOffice, iar codul din spatele acestor interfete a fost scris in limbajul de programare OpenOffice Basic.

Principalele avantaje ale acestei solutii sunt:

- portabilitatea: interfata grafica si codul Basic sunt specificate in fisiere XML si copierea lor de pe un sistem pe altul este foarte facila, si deasemeni ruleaza fara modificari pe orice alt sistem atat timp cat se foloseste aceeasi versiune de OpenOffice, si anume 1.1.4

- usurinta crearii si testarii acestor interfete (compilarea codului se face la rulare, astfel ca operatiile de debug s-au efectuat foarte usor, nefiind necesara o compilare a acestora); deasemeni pentru creareai interfetei grafice OpenOffice pune la dispozitie un designer de ferestre, setarea proprietatilor componentelor ferestrelor si asocierea de evenimente fiind o joaca de copii (cu facultate)

4.4 Implementarea functiilor aplicatiei

4.4.1 Operatia de semnare

Date de intrare:

- documentul curent

- un fisier P12 si parola de acces la acesta

Date de iesire:

- un fisier salvat in aceeasi locatie ca fisierul original, la care se adauga extensia suplimentara p7s

Erori posibile:

- fisierul original este stocat pe un mediu de citire read-only

- parola de acces la fisierul P12 este invalida

Scriptul care efectueaza operatia de semnare este:

~/.securedoc/sign_file.sh

Acesta citeste datele de intrare din fisierele:

~/.securedoc/sign_input_file = numele fisierului original

~/.securedoc/signed_file_name = numele fisierului semnat

~/.securedoc/profile_p12 = numele fisierului P12 utilizat pentru semnare

~/.securedoc/password = parola de acces la fisierul P12

Comanda OpenSSL pentru semnare este:

```
openssl smime -sign -inform DER -outform DER -binary -nodetach
   -in <nume_fisier_de_semnat>
   -out <nume_fisier_semnat>
   -signer <certificat_utilizator>
   -passin pass:<parola_p12>
```

4.4.2 Operatia de verificare semnatura

Date de intrare:

- numele documentului curent

Date de iesire:

- fisierul cu certificatele utilizatorilor care au semnat documentul

- fisierul original, folosit pentru a fi incarcat in OpenOffice

Erori posibile:

- fisierul original sa nu fie in format SMIME, astfel neputand fi decodificat de catre OpenSSL

Scriptul care efectueaza operatia de extragere a semnaturilor este:

~/.securedoc/extract_signatures.sh

Acesta citeste datele de intrare din fisierul:

~/.securedoc/signed_file_name = numele fisierului semnat

Operatia OpenSSL pentru extragerea semnaturilor este:

```
openssl smime –verify -inform DER -outform DER -binary
   -in <nume_fisier_semnat>
   -out <fisier_original_extras>
   -signer <lista_certificate_extrase>
```

4.4.3 Operatia de criptare

Date de intrare:

- documentul curent

- lsita recipientilor pentru criptare

Date de iesire:

- un fisier salvat in aceeasi locatie ca fisierul original, la care se adauga extensia suplimentara p7m

Erori posibile:

- fisierul original este stocat pe un mediu de citire read-only

Scriptul care efectueaza operatia de criptare este:

~/.securedoc/crypt_file.sh

Acesta citeste datele de intrare din fisierele:

~/.securedoc/crypt_input_file = numele fisierului original

~/.securedoc/crypted_file_name = numele fisierului semnat

~/.securedoc/crypt_recipients = lista certificatelor folosite pentru criptare

Comanda OpenSSL pentru criptare este:

```
openssl smime -encrypt -inform DER -outform DER -binary -nodetach
   -in <nume_fisier_de_criptate>
   -out <nume_fisier_criptat>
   <lista_fisierelor_cu_certificatele_recipientilor>
```

4.4.4 Operatia de decriptare

Date de intrare:

- numele documentului curent

- parola de acces la fisierul P12

Date de iesire:

- fisierul original, folosit pentru a fi incarcat in OpenOffice

Erori posibile:

- parola de acces la fisierul P12 sa fie invalida

- fisierul original sa nu fie in format SMIME, astfel neputand fi decriptat de catre OpenSSL

Scriptul care efectueaza operatia de decriptare este:

~/.securedoc/decrypt_file.sh

Acesta citeste datele de intrare din fisierul:

~/.securedoc/crypted_file_name = numele fisierului criptat

Operatia OpenSSL pentru decriptare este:

```
openssl smime -decrypt -inform DER -outform DER -binary
   -in <nume_fisier_criptat>
   -out <fisier_original_extras>
   -recip <certificatul_extras_din_p12>
   -inkey <cheia_privata_extrasa_din_p12>
```

4.4.5 Management repository

Operatiile de management repository consta in copierea sau stergerea de certificate sau fisiere P12 din cele patru repository-uri.

Nota: pentru repository-urile Trusted Root CA si Trusted Intermediate CA utilizatorul trebuie sa aibe drepturi de administrator in sistem.

Trusted Root CA

Pentru a instala manual un certificat de RootCA se va proceda astfel:

- se copie fisierul cu certificatul respectiv in format PEM in directorul `/etc/ssl/certs` (exemplu: ca.crt)

- se genereaza un hash al certificatului pe 32 biti

- se creeaza un link simbolic in acelasi director cu numele fiind hash-ul generat la pasul precedent si extensia ".0" catre fisierul ca.crt.

Exemplu: presupunem ca avem fisierul ca.crt in directorul `/etc/ssl/certs`, atunci, pentru ca acesta sa fie instalat ca si certificat de RootCA se vor rula rumatoarele comenzi:

```
/etc/ssl/certs # openssl x509 -noout -hash -in ca.crt
3acc0086
/etc/ssl/certs # ln -s ca.crt 3acc0086.0
```

Trusted Intermediate CA

Pentru a instala manual un certificat de SubCA se va proceda astfel:

- se copie fisierul cu certificatul respectiv in format PEM in directorul `/etc/ssl/certs` (exemplu: subca.crt)

- se genereaza un hash al certificatului pe 32 biti

- se creeaza un link simbolic in acelasi director cu numele fiind hash-ul generat la pasul precedent si extensia ".0" catre fisierul subca.crt.

Exemplu: presupunem ca avem fisierul subca.crt in directorul `/etc/ssl/certs`, atunci, pentru ca acesta sa fie instalat ca si certificat de SubCA se vor rula rumatoarele comenzi:

```
/etc/ssl/certs # openssl x509 -noout -hash -in subca.crt
bc4409fe
/etc/ssl/certs # ln -s subca.crt bc4409fe.0
```

Other People

Pentru a instala manual un certificat de End Entity se va proceda astfel:

- se copie fisierul cu certificatul respectiv in format PEM in directorul `~/.securedoc/repository/entities` (exemplu: ee.crt)

- se genereaza un hash al certificatului pe 32 biti

- se redenumeste fisierul ee.crt in numele generat de hash la care se adauga extensia ".crt"

Exemplu: presupunem ca avem fisierul ee.crt in directorul `~/.securedoc/repository/entities`, atunci pentru ca acesta sa fie instalat ca si certificat in repository-ul Other People se vor rula rumatoarele comenzi:

```
/etc/ssl/certs # openssl x509 -noout -hash -in ee.crt
99feea0b
/etc/ssl/certs # mv ee.crt 99feea0b.crt
```

Personal

Pentru a instala manual un fisier P12 in repository-ul Personal se va proceda astfel:

- se copie fisierul P12 respectiv in directorul
`~/.securedoc/repository/identities`

4.5 Comunicarea interfata-scripturi aplicatie

Comunicarea intre codul din spatele interfetei grafice si scripturile din directorul unde s-a instalat aplicatia se face utilizand fisiere temporare cu mesaje.

Astfel, procesul de desfasurare al unei operatii intiata din interfata grafica decurge astfel:

- se culeg datele din interfata (precum ghioceii primavara)
- se scriu in fisiere temporare in directorul ~/.securedoc/tmp
- se ruleaza scriptul corespunzator operatii dorite
- acesta face prelucrarile pentru care a fost programat si scrie rezultatele de iesire in fisiere temporare in acelasi director temporar

Pentru a preveni umplerea acestui director temporar cu date inutile (si senzitive dealtfel, caci si parolele pentru operatiile de acces la fisierele P12 sunt transmise tot ca si fisiere temporare din acest director) dupa fiecare rulare a scriptului fisierele sunt sterse astfel:

- fisierele de intrare sunt sterse la terminarea executiei scriptului
- fisierele de iesire sunt sterse dupa ce sunt citite de codul din interfata

Capitolul 5: Manual de utilizare

5.1 Cerinte sistem

Pentru a instala aplicatia pe un sistem, acesta trebuie sa aiba preinstalate urmatoarele programe:

OpenOffice versiunea 1.1.4. Acesta este platforma pe care va rula modulul SecureDoc, si reprezinta o alternativa freeware si open-source pentru aplicatiile de procesare documente (de genul, dar nu dupa filozofia Microsoft Office). De ce am ales aceasta platforma pentru dezvoltarea aplicatiei ? In primul rand datorita documentatiei existente si a disponibilitatii surselor OpenOffice, dar si datorita faptului ca pentru aceasta suita office nu exista un modul flexibil si usor de utilizat pentru implementarea functiilor de semnare si criptare electronica a documentelor.

Aceasta aplicatie poate fi downloadata de pe site-ul www.openoffice.org. Versiunea care a fost utilizata pentru testele prezente in acest document este 1.1.4 pentru sistemul de operare Linux. Deasemeni, cu mici modificari (adaptari), modulul SecureDoc poate fi rulat si pe versiunea OpenOffice pentru sistemul de operare Windows.

OpenSSL v0.9.7g. Reprezinta o suita de aplicatii freeware si open-source care ofera o gama variata de comenzi pentru efectuarea de operatii criptografice. Este disponibila pentru aproape orice sistem de operare existent in prezent, este usor de utilizat si poate fi integrata in terte aplicatii printr-o varietate de metode.

OpenSSL poate fi downloadat de pe site-ul www.openssl.org.

Java JRE 1.4. Ofera runtime-ul necesar rularii modulelor OpenOffice dezvoltate in limbajul Java. Pentru adaugarea de extensii OpenOffice, dezvoltatorii acestei suite ofera un SDK care permite utilizatorilor sa programeze module folosind in limbajul Java sau C++. Pentru acest proiect am ales limbajul Java datorita usurintei in dezvoltare (criteriu determinant in cazul in care raportul timp / facilitati dezvoltate este mic).

5.2 Instructiuni de instalare

Pentru instalarea si configurarea optima a aplicatiilor se vor executa pasii urmatori:

1) se va instala masina virtuala Java (JRE); aceasta operatie se face doar daca pe sistem nu este deja instalata aceasta aplicatie;

2) se instaleaza aplicatie OpenOffice 1.1.4, decomprimand arhiva cu kitul downloadat de pe internet, si ruland aplicatia./setup, care va cere utilizatorului datele de configurare ale aplicatiei; pentru exemplificare, directorul de instalare se va alege `/opt/ooo-1.1.4`;

3) dupa instalarea OpenOffice, se va rula aplicatia;

`/opt/ooo-1.1.4/program/jvmsetup`

care va invita politicos utilizatorul sa specifice calea catre masina virtuala Java care va fi integrata cu OpenOffice;

4) se va instala modulul SecureDoc, astfel: se vor copia, din directorul ce contine kitul acestuia, fisierele din directorul ./ooo-1.1.4 in directorul in care este instalat OpenOffice (astfel, continutul directorul `./user` va fi copiat in directorul `/opt/ooo-1.1.4`, suprascriind fisierele existente in sistem); apoi, pentru fiecare utilizator in parte, se vor copia fisierele din directorul `./home` din kit in directoarele home ale fiecarui user in parte.

5.3 Management repository

Aplicatia este formata din mai multe module, cel mai cuprinzator dintre acestea fiind componenta de management a certificatelor digitale.

Deoarece pentru operatiile criptografice am utilizat functiile oferite de OpenSSL, trebuie sa explicam modul de organizare al acestei aplicatii. OpenSSL foloseste un repository global disponibil tuturor utilizatorilor sistemului, care contine in special (si este recomandat sa contina numai) certificatele de RootCA si SubCA. Directorul in care sunt stocate aceste certificate, pe sistemul de operare Linux, este:

`/etc/ssl/certs`

Acesta vine preinstalat cu certificatele unor autoritati publice binecunoscute (cum ar fi: Thawte, VeriSign). Formatul in care sunt stocate fisierele in acest director este urmatorul:

- fisierele de forma `hhhhhhhh.0` reprezinta certificate (linkuri catre fisiere ce contin certificate) care formeaza repository-ul public al OpenSSL

- fisierele cu orice alt nume sunt ignorate.

Ce reprezinta `hhhhhhhh.0` si cum se calculeaza ? Grupul de 8 litere hexa reprezinta un hash pe 32 biti al certificatului respectiv, calculat cu comanda:

```
openssl x509 -hash -noout -in <fisier_certificat>
```

si extensia data de cifra zero reprezinta un filtru de cautare in acest director de catre OpenSSL.

Functiile oferite de componenta management repository sunt urmatoarele:

- management autoritati RootCA
- management autoritati SubCA
- management certificate End Entity ale tertilor utilizatori
- management certificate si chei personale

Accesul la aceste functii se face accesand iconita Repository Management din bara cu butoane a aplicatiei.

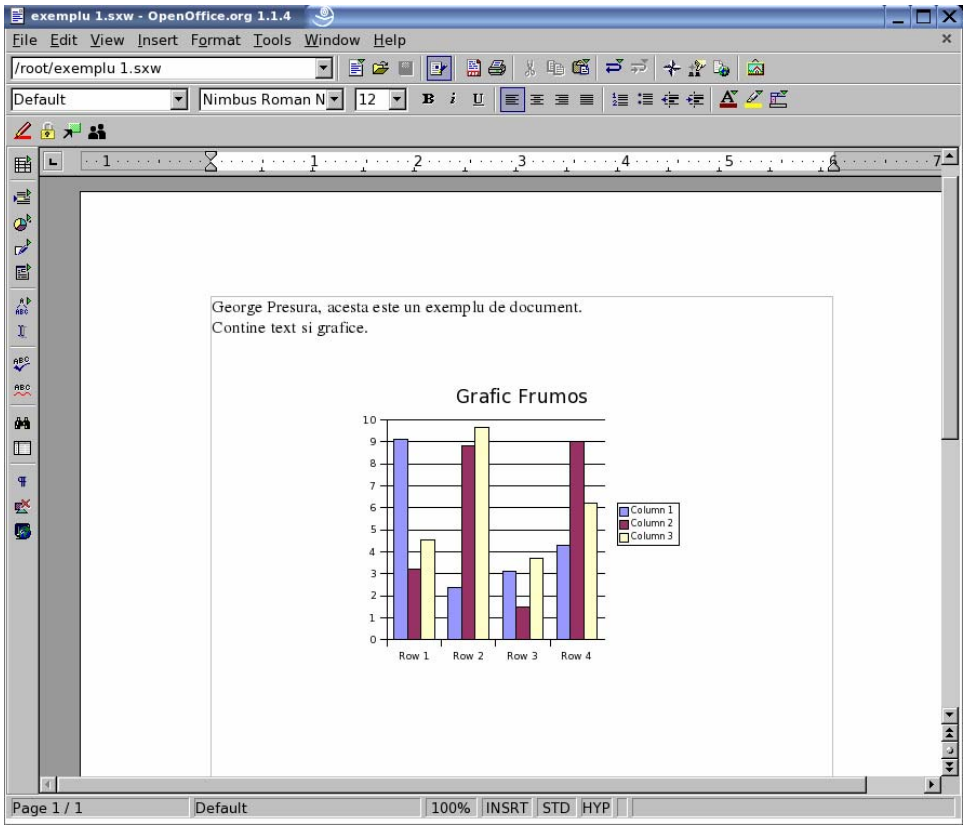

Figure 1 Fereastra aplicatiei (se observa bara cu cele 4 butoane ale modulului)

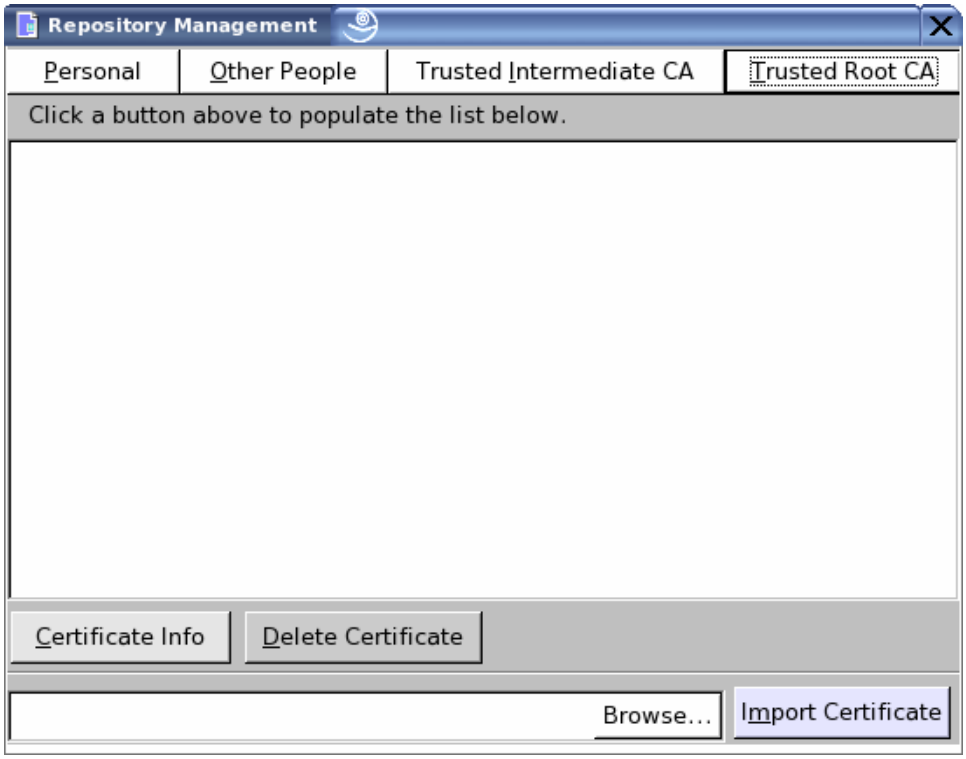

Figure 2 Fereastra Management Repository

5.3.1 Management autoritati RootCA

Repository-ul RootCA contine certificatele de root, autosemnate. Lista acestora este compusa din certificatele din repository-ul public al OpenSSL, care indepliesc conditia ca Subject-ul si Issuer-ul sa fie identice.

Listarea acestor certificate se face apasand butonul Trusted Root CA din fereastra Repository Management.

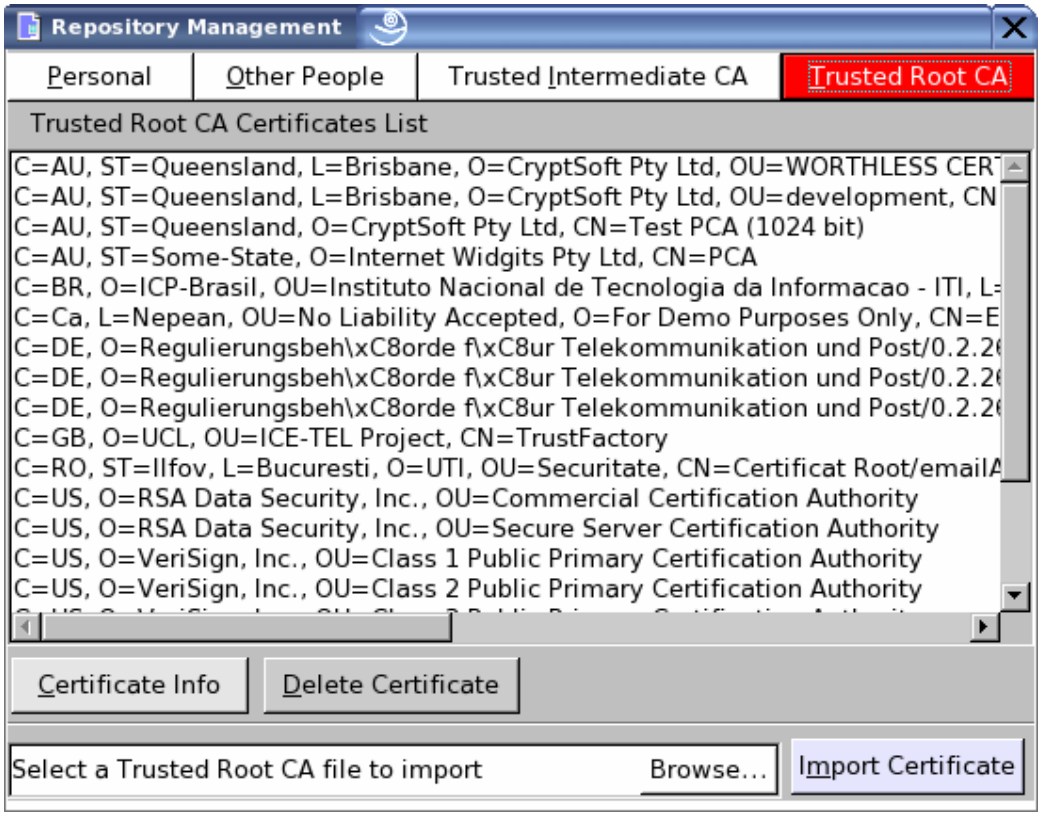

Figure 3 Lista certificatelor autoritatilor RootCA

5.3.2 Management autoritati SubCA

Repository-ul SubCA contine certificate ale autoritatilor de certificare intermediare. Lista acestora este compusa din certificatele din repository-ul public al OpenSSL, care indepliesc conditia ca Subject-ul si Issuer-ul sa fie diferite. Nota: este posibil ca in repository-ul public al OpenSSL sa fie certificate de End Entity, si in acest caz ele vor apare ca si certificate de autoritati de certificare intermediare, lucru care trebuie evitat, astfel ca operatiile de management a repository-ului OpenSSL sunt permise numai utilizatorilor cu drepturi de administrator in sistem.

Listarea acestor certificate se face apasand butonul Trusted Intermediate CA din fereastra Repository Management.

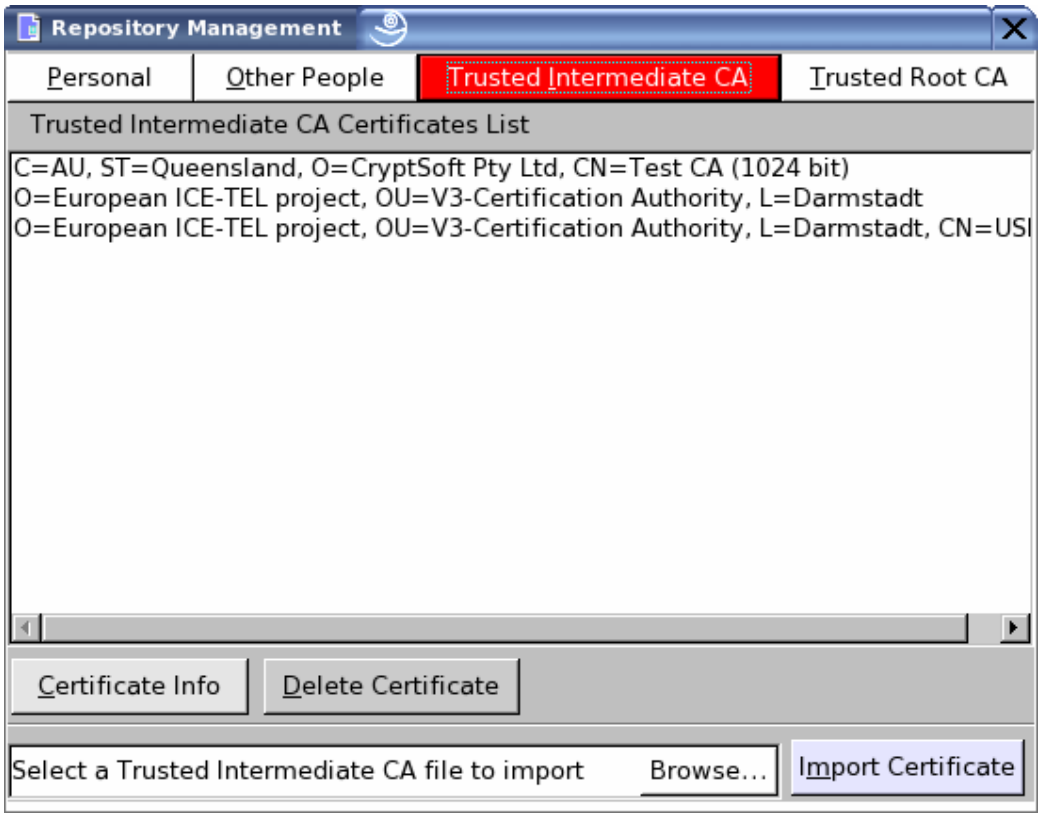

Figure 4 Lista certificatelor autoritatilor SubCA

5.3.3 Management certificate End Entity ale tertilor utilizatori

Acesta este un repository privat, care este gestionat de fiecare utilizator din sistem in parte, si contine certificatele personelor cu care utilizatorul respectiv a facut schimb de mesaje criptate / semnate. Certificatele din acest repository au in special rolul de a fi utilizate pentru a trimite mesaje criptate catre alti utilizatori (posesorilor cheilor private care fac pereche si casa de piatra cu cheia publica a acestor certificate).

Adaugarea de certificate se face atat manual, cat si utilizand facilitatea de a extrage un certificat dintr-un mesaj semnat primit de la un tert utilizator.

Listarea acestor certificate se face apasand butonul Other People din fereastra Repository Management.

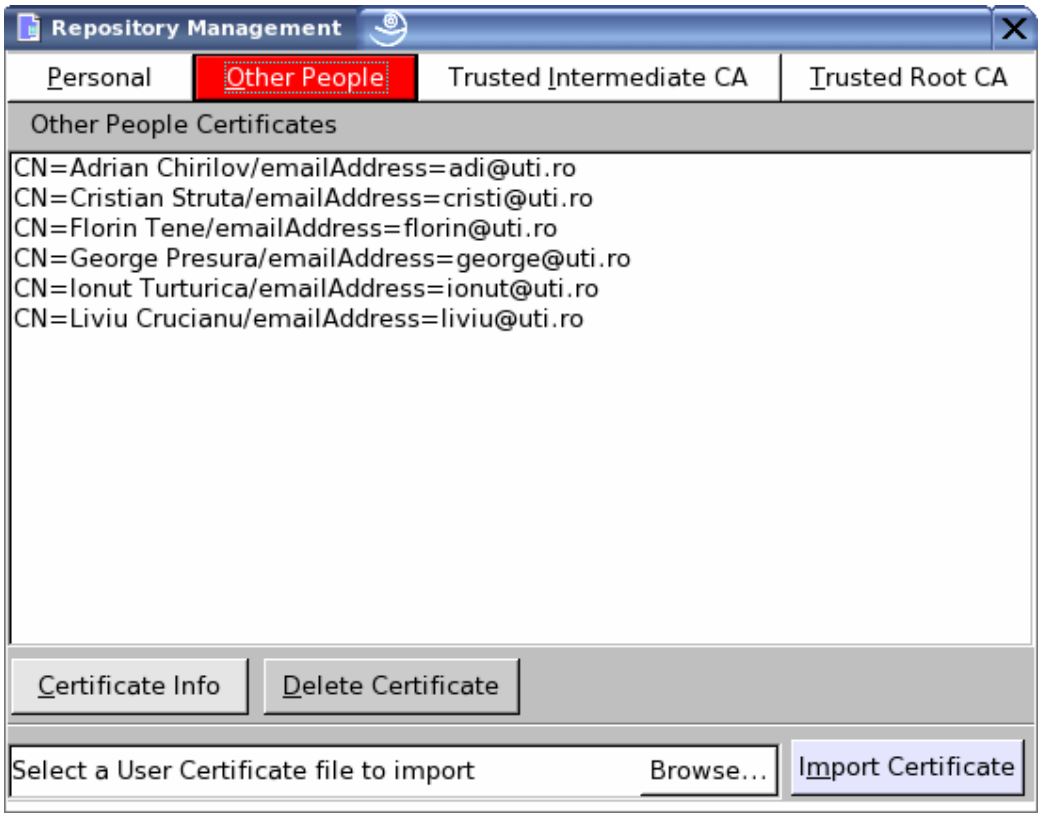

Figure 5 Lista certificatelor tertilor utilizatori

5.3.4 Management certificate si chei private

Acesta este un repository privat, care este gestionat de fiecare utilizator din sistem in parte, si contine o lista de fisiere cu extensia P12. Aceste fisiere contin date codificate in formatul PKCS#12, care impacheteaza intr-un singur fisier un certificat de utilizator si cheia privata corespunzatoare acestuia. Cheia privata este protejata de parola. Accesul la acest director trebuie restrictionat altor utilizatori, si fisierele P12 trebuie sa fie protejate de parola.

Listarea fisierelor P12 din acest repository se face apasand butonul Personal din fereastra Repository Management.

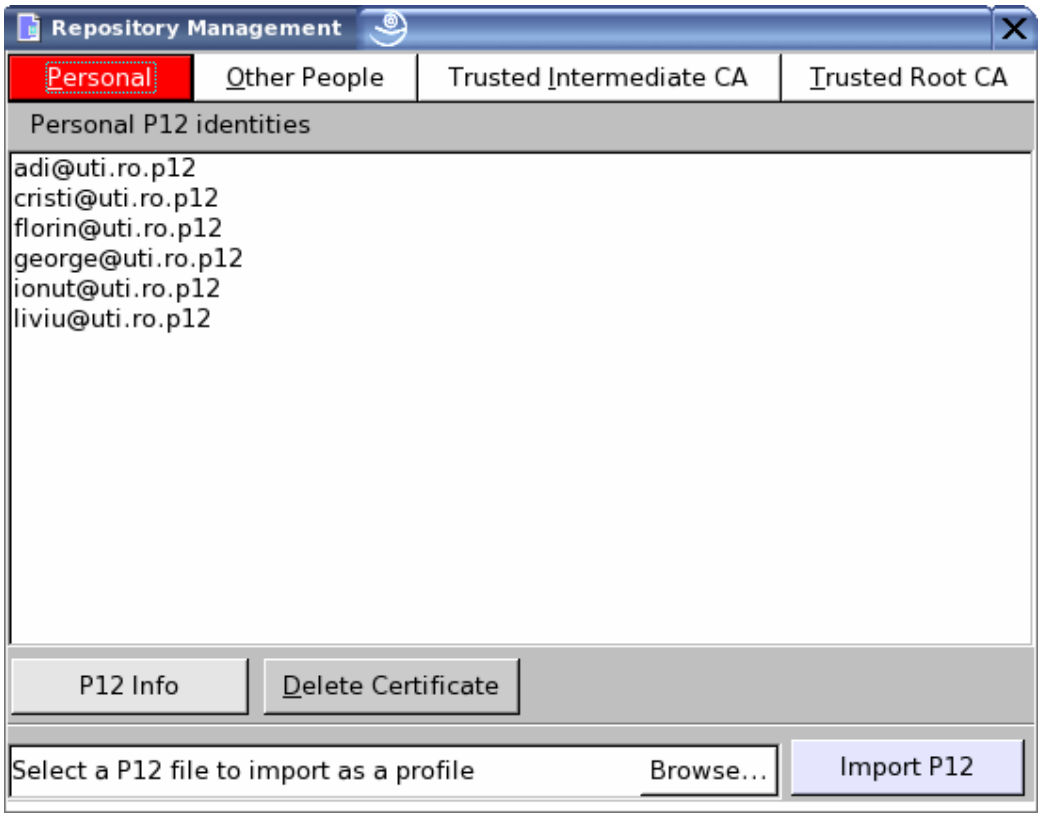

Figure 6 Lista fisierelor P12 (identitatile = certificatele personale)

5.3.5 Stergerea unui certificat/P12 din repository

Pentru a sterge un certificat/P12 din repository se selecteaza certificatul/P12 respectiv apoi se apasa butonul Delete Certificate. Inainte de a fi sters fisierul respectiv, utilizatorului i se va cere o confirmare a operatiei de stergere.

In cazul certificatelor autoritatilor de RootCA si SubCA, utilizatorul trebuie sa aiba drepturi de acces (scriere) asupra directorul ce contine repository-ul OpenSSL (in cadrul sistemelor de operare Linux utilizatorul trebuie sa fie root, iar in cazul sistemelor Windows NT utilizatorul trebuie sa faca parte din grupul Administrators).

Atentie: inainte de a sterge un fisier P12 din repository, utilizatorul trebuie sa aiba o copie de siguranta a acestuia, si deasemeni sa salveze parola de acces la fisier. Aceastea sunt necesare in cazul in care anumite documente au fost criptate cu certificatul din fisierul P12, iar accesul la acestea va fi necesar in viitor.

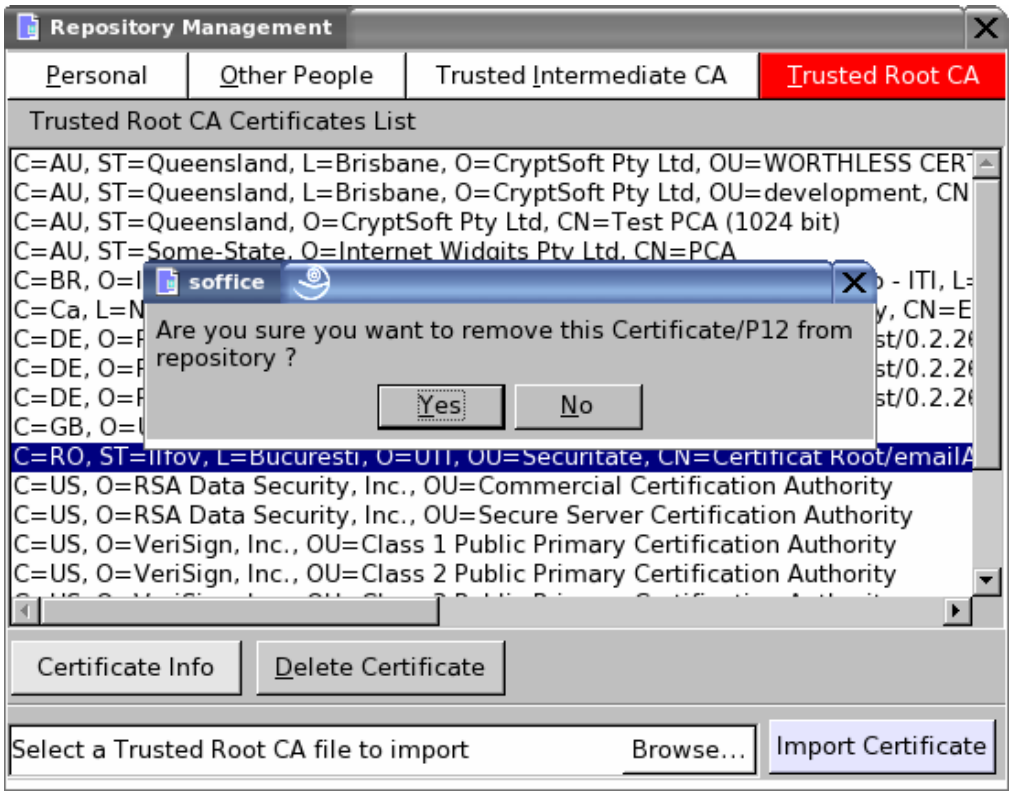

Figure 7 Confirmare pentru stergere certificat/P12 dintr-un repository

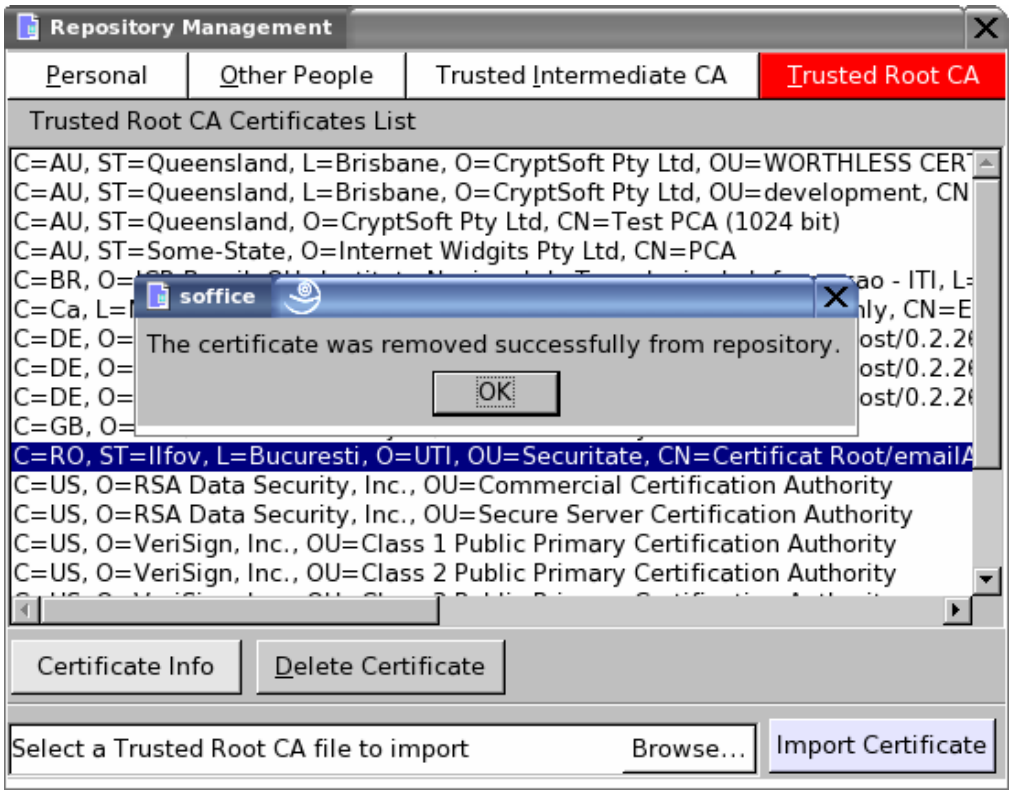

Figure 8 Mesaj de confirmare a stergerii certificatului din repository

5.3.6 Importul unui certificat/P12 in repository

Pentru a importa un certificat/P12 in repository, utilizatorul va selecta repository-ul dorit apoi va apasa butonul Browse pozitionat in partea inferioara a ferestrei Repository Management, apoi va selecta de pe disc fisierul pe care doreste sa il importe, apoi va apasa butonul Import Certificate.

Nota: fisierele care contine certificate trebuie sa fie in format PEM, iar fisierele P12 trebuie sa fie in format DER.

Pentru a importa un certificat in repository-ul Trusted Root CA sau Trusted Intermediate CA, utilizatorul trebuie sa aiba drepturi de acces asupra repository-ului public al OpenSSL (vezi detaliile de mai sus).

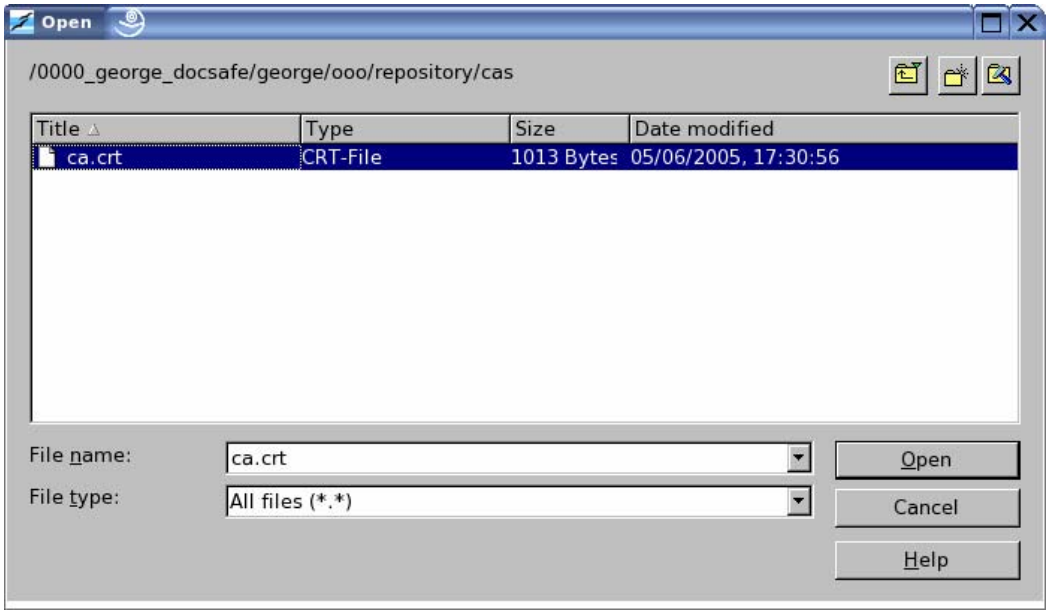

Figure 9 Selectare fisierului cu certificatul/P12-ul pentru import in repository

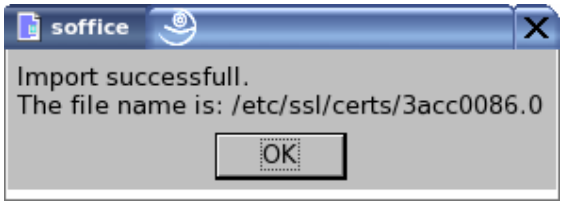

Figure 10 Mesaj de confirmare a operatiei de import (contine si numele fisierului in care a fost copiat certificatul/P12-ul)

5.3.7 Afisare detalii despre un certificat

Afisarea operatiilor despre un certificat (uite cum m-am repetat) se va face intr-o fereastra care contine doua zone de text. Cea superioara contine detalii despre

certificat, in format human-readable, iar cea inferioara contine detalii despre starea de valabilitate a certificatului.

Pentru extragerea informatiilor din fisierul care contine certificatul s-a utilizat comanda OpenSSL: x509, iar pentru operatia de verificare a certificatului s-a utilizat comanda OpenSSL: verify.

Pentru a determina daca un certificat este valid sau nu, se vor efectua urmatoarele operatii de catre OpenSSL:

- se verifica daca data sistemului se incadreaza in data de valabilitate a certificatului (deci utilizatorul trebuie sa aiba data sistemului cat mai apropiata de data reala);

- se verifica daca autoritatea care a semnat certificatul respectiv se gaseste in repository-ul public al OpenSSL;

- NU se verifica starea de revocare a unui certificat.

Pe baza acestor verificari putem trage (cu incredere) concluzia ca pentru ca un certificat sa fie valid pentru aplicatie, acesta trebuie sa respecte conditiile de date si sa existe fie in repository-ul Trusted Root CA sau Trusted Intermediate CA certificatul autoritatii de certificare care l-a semnat. Astfel, in cadrul unei organizatii care implementeaza o intrastructura PKI privata cu propriile autoritati de certificare, adminitratorii fiecarui sistem trebuie sa instaleze certificatele autoritatilor de certificare in repository-ul OpenSSL.

In cazul in care un certificat nu este valid, se afiseaza in fereastra inferioara motivul pentru care certificatul nu este valid. Acest mesaj reprezinta rezultatul executiei operatiei de verificare a OpenSSL.

Nota: este recomandata verificarea periodica a certificatelor de End Entity si stergerea acestora din repository in cazul in care acestea expira. Deasemeni, administratorii de sistem trebuie sa verifice periodic starea certificatelor de RootCA si SubCA si inlocuirea / instalarea de noi certificate in cazul in care acestea expira, pentru a asigura consistenta infrastructurii PKI.

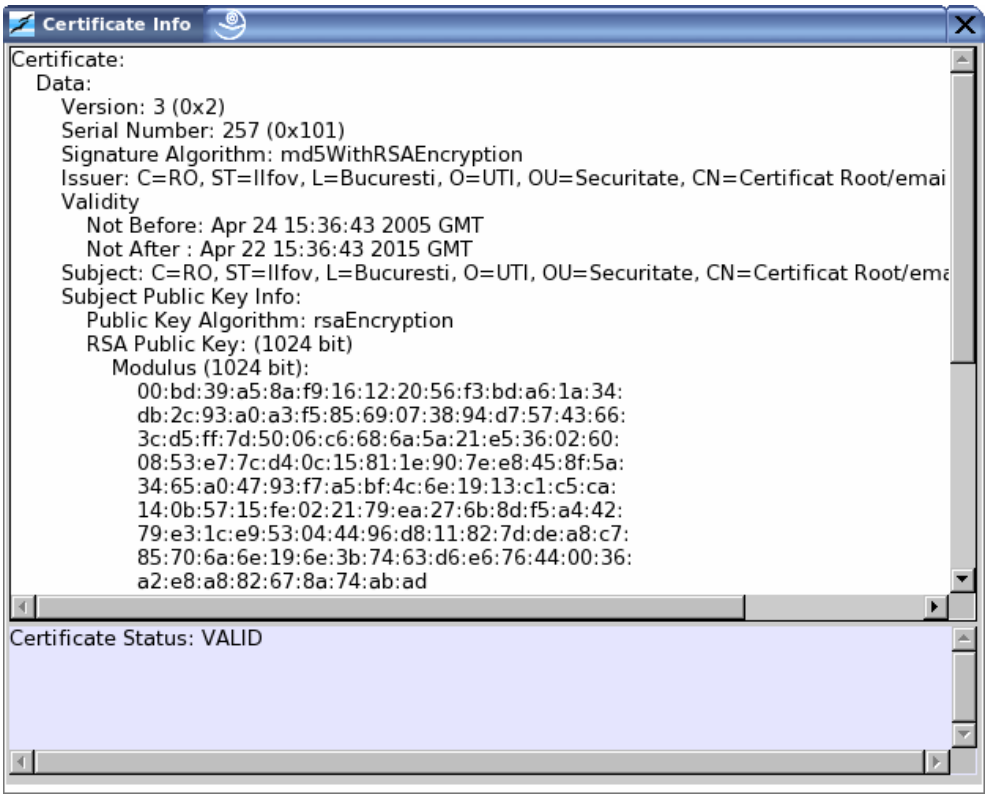

Figure 11 Exemplu de certificat valid

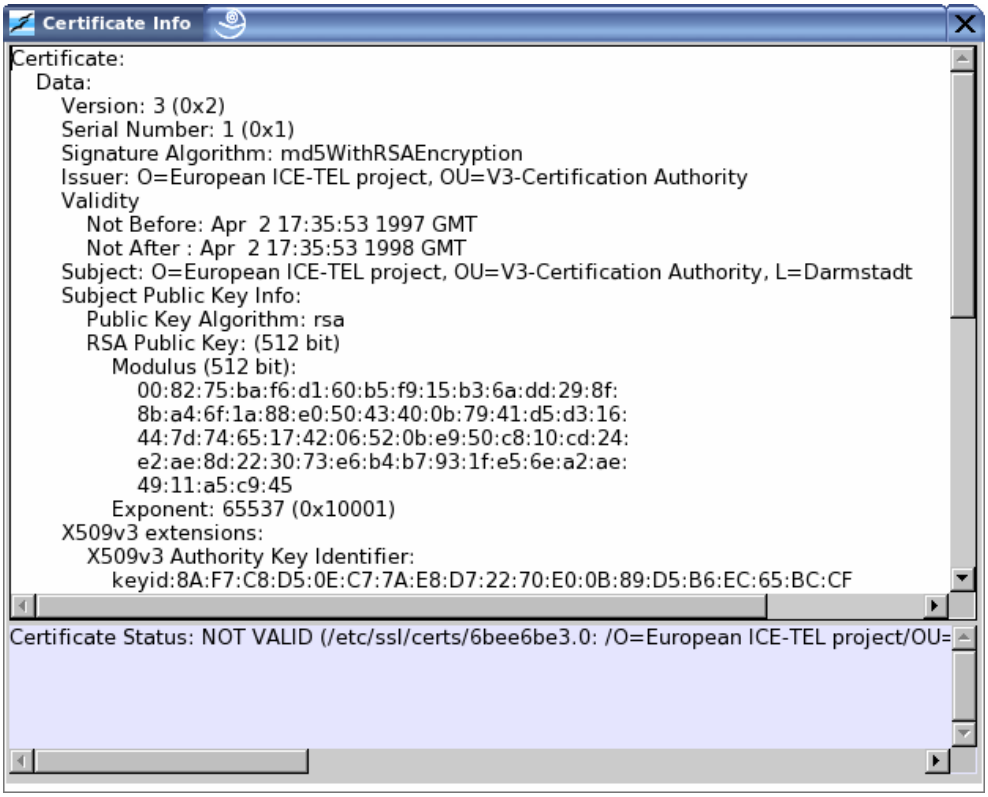

Figure 12 Exemplu de certificat invalid (in acest caz este expirat)

5.3.8 Afisare detalii despre un fisier P12

Pentru a afisa detali despre un certificat P12, trebuie ca utilizatorul sa introduca in plus parola de acces la fisierul P12.

Figure 13 Introducere parola pentru acces la fisierul P12

5.4 Semnare documente

Operatia de semnare a unui document presupune crearea unui fisier cu acelasi nume ca si cel original la care se adauga extensia suplimentara p7s.

Etapele semnarii unui document sunt urmatoarele:

- utilizatorul creaza un nou document sau deschide unul existent;
- salveaza documentul respectiv;
- apeleaza optiunea de semnare a documentului, prin accesarea butonului corespunzator din toolbar-ul aplicatiei;
- selecteaza o identitate din lista (fisier P12);
- introduce parola de acces la fisierul P12 corespunzator identitatii selectate;
- primeste mesaj de confirmare daca operatia de semnare a decurs cu succes sau au aparut erori.

Cele mai comune erori care pot apare in acest caz sunt: introducerea unei parola gresite pentru fisierul P12 sau incercarea de semnare a unui document de pe un mediu read-only.

Pentru a semna un document, utilizatorul are nevoie de un certificat digital si cheia privata corespunzatoare cheii publice din certificat. Operatia de semnare consta in crearea unui fisier cu extensia p7s (care provine de la PKCS#7 Signed Message) al carui continut este format din fisierul original (documentul in clar), certificatul

utilizatorului care semneaza documentul si o semnatura. Directorul in care va fi salvat acest fisier este acelasi cu directorul in care se afla fisierul original, iar numele acestuia este format din numele (inclusiv extensia) fisierului original la care se adauga extensia suplimentara p7s. Astfel, in cazul in care se doreste semnarea unui fisier care se afla intr-un director in care utilizatorul nu are acces pentru scriere, atunci acesta trebuie sa copie fisierul intr-un director temporar, sa deschida si sa semneze fisierul din acea locatie.

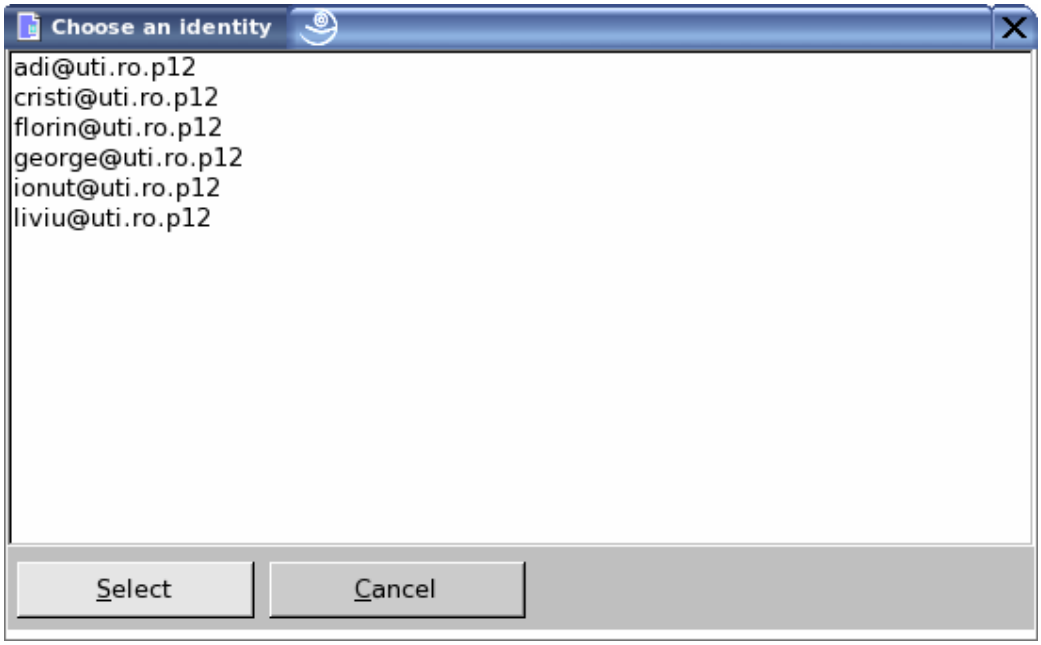

Figure 14 Fereastra pentru alegerea profilului utilizat la semnarea documentului

Figure 15 Confirmare semnare document

Figure 16 Introducerea parolei de acces la fisierul P12

Figure 17 Mesaj de confirmare a crearii unui fisier semnat

5.5 Criptare Documente

Operatia de criptare a unui document presupune crearea unui fisier cu acelasi nume ca si cel original la care se adauga extensia suplimentara p7m.

Etapele criptarii unui document sunt urmatoarele:

- utilizatorul creaza un nou document sau deschide unul existent;

- salveaza documentul respectiv;

- apeleaza optiunea de criptare a documentului, prin accesarea butonului corespunzator din toolbar-ul aplicatiei;

- selecteaza lista de certificate pentru care doreste sa cripteze documentul;

- primeste mesaj de confirmare daca operatia de criptare a decurs cu succes sau au aparut erori.

O eroare comuna care poate apare pe parcursul operatiei de criptare o reprezinta incercarea de criptare a unui document de pe un mediu read-only.

Pentru a cripta un document, utilizatorul trebuie sa specifice o lista de recipienti pentru care se va cripta documentul respectiv.

Operatia de criptare se efectueaza astfel: se ia documentul in clar si se cripteaza cu o cheie simetrica generata aleator. Apoi cheia simetrica este criptata pe rand cu cheile publice ale certificatelor selectate pentru criptare. Din aceste doua structuri se genereaza un fisier cu extensia p7m (care provine de la PKCS#7 Crypted Message).

Directorul in care va fi salvat acest fisier este acelasi cu directorul in care se afla fisierul original, iar numele acestuia este format din numele (inclusiv extensia) fisierului original la care se adauga extensia suplimentara p7m. Astfel, in cazul in care se doreste semnarea unui fisier care se afla intr-un director in care utilizatorul nu are acces pentru scriere, atunci acesta trebuie sa copie fisierul intr-un director temporar, sa deschida si sa semneze fisierul din acea locatie.

In cazul criptarii unui document, acesta poate fi decriptat numai de catre cei care au acces la cheia privata a certificatelor din lista de recipienti cu care a fost criptat documentul. O situatie oarecum amuzanta, dar nu e de ras, ba chiar de plas cu lacrimi de crocodil atunci cand te lovesti de ea, este ca utilizatorul sa specifice in lista de recipienti anumite certificate, dar nu si (unul dintre) certificatele sale. Astfel, in cazul in care utilizatorul va sterge fisierul dupa ce l-a criptat, este posibil ca acesta sa nu il mai poata deschide. Este deci recomandat ca utilizatorul sa se adauge si pe el (unul din certificatele din identitatile P12) in lista de recipienti atunci cand cripteaza un document.

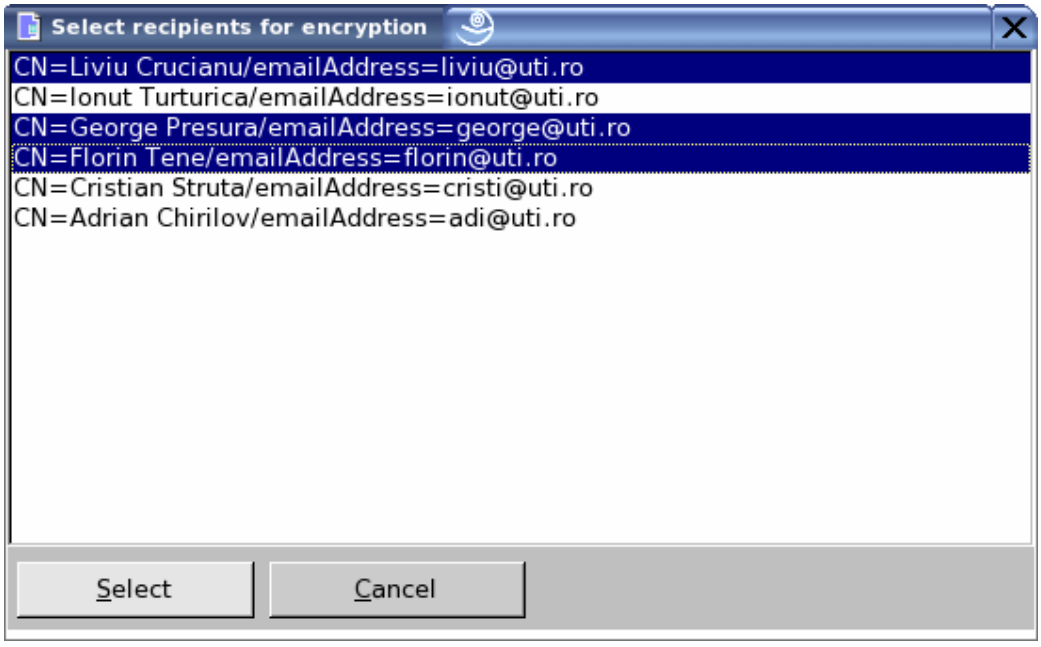

Figure 18 Alegerea recipientilor pentru criptarea unui document

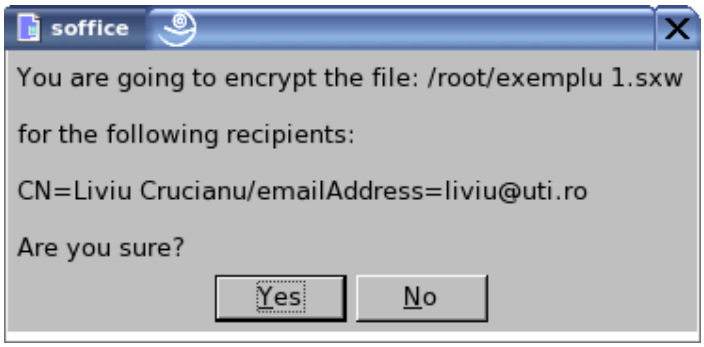

Figure 19 Confirmare pentru operatia de criptare

Figure 20 Mesaj de succes pentru operatia de criptare (se afiseaza si numele noului fisier criptat)

5.6 Dezinstalare aplicatie

Dezinstalarea aplicatiei se poate face in doua moduri: dezinstalarea pentru un utilizator sau dezinstalarea din suita OpenOffice.

Dezinstalarea pentru un singur utilizator presupune stergerea fisierelor din directorul `~/.securedoc`. Atentie: inainte de a dezinstala aplicatia, este recomandat ca utilizatorul sa salveze fisierele P12, doarece acestea contin cheile private care ar putea fi necesare pentru decriptarea unor documente primite criptat de la alti utilizatori.

Dezinstalarea globala din aplicatia OpenOffice se face astfel:

- se va opri aplicatia OpenOffice
- se vor sterge fisierele din urmatoarele directoare:

`/opt/ooo-1.1.4/user/basic/Standard`

`/opt/ooo-1.1.4/user/config/soffice.cfg`

(unde `/opt/ooo-1.1.4/` se va inlocui cu directorul in care este instalat OpenOffice pe sistemul respectiv)

Capitolul 6: Bibliografie

Bibliografie

1. OpenOffice versiunea 1.1.4
   www.openoffice.org
2. OpenSSL
   www.openssl.org
3. Linux
   www.linux.org
4. Google
   www.google.com
5. Underestanding Public-Key Infrasctructure
   Carlisle Adams, Steve Lloyd

Capitolul 7: Anexe

7.1 securedoc_sign

```
Dim sFile
Dim sIdentity
Dim oDialog
Dim PROGRAM_FOLDER As String
Dim PROGRAM_REPOSITORY As String
Dim PROGRAM_REPOSITORY_ENTITIES As String
Dim PROGRAM_REPOSITORY_IDENTITIES As String
Dim PROGRAM_TMP As String

Sub Main

initVariables()

sFile = ThisComponent.getLocation()
sIdentity = ""
BasicLibraries.LoadLibrary("Tools")
oDialog = LoadDialog("Standard", "dlgChooseIdentity")

' check whatever the document is saved and if it's name really exists
if sFile <> "" and FileExists(sFile) then
      ' preprocess the file name: remove the "file://" prefix
      sFile = ConvertFromURL(sFile)
      ' choose an identity from the list
      getPersonalCertificates()
      oDialog.Execute()
      ' ask for confirmation
      if sIdentity <> "" then
            if MsgBox("You are going to sign the file: " & sFile &
Chr(10) & Chr(10) & "with the P12 identity: " & sIdentity & Chr(10) &
Chr(10) & "Are you sure?", 4) <> 6 then
                  exit sub
            else
                  ' ask for the P12 password
                  pass = ""
                  pass_in  = InputBox("Please enter the P12
password", "P12 Password", "")
                  ' write the password to the file
                  aFile   = PROGRAM_tMP & "/password"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, pass_in
                  Close #iNumber
                  ' write file name
                  aFile   = PROGRAM_TMP & "/sign_input_file"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sFile
                  Close #iNumber
                  aFile   = PROGRAM_TMP & "/signed_file_name"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sFile & ".p7s"
                  Close #iNumber
                  aFile   = PROGRAM_TMP & "/profile_p12"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sIdentity
```

```
                  Close #iNumber
                  ' run the external script
                  Shell (PROGRAM_FOLDER & "/sign_file.sh", 6, "",
true)
                  ' read the script output
                  aFile   = PROGRAM_TMP & "/sign_result"
                  iNumber = FreeFile()
                  Open aFile For Input Access Read As #iNumber
                  sResult = ""
                  While not eof(#iNumber)
                        Line Input #iNumber, sLine
                        sResult = sResult & sLine & Chr(13) & Chr(10)
                  wend
                  Close #iNumber
                  MsgBox sResult
            end if
      end if
else
      MsgBox "The file must be saved before you may sign it!"
end if

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/password")
Kill(PROGRAM_TMP & "/profile_p12")
Kill(PROGRAM_TMP & "/sign.pem")
Kill(PROGRAM_TMP & "/sign_input_file")
Kill(PROGRAM_TMP & "/signed_file_name")
Kill(PROGRAM_TMP & "/sign_result")

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub getPersonalCertificates

initVariables()

oListBox = oDialog.GetControl("lstCertificates")
oListBox.removeItems(0, oListBox.getItemCount())

' run the script to generate P12 list
Shell (PROGRAM_FOLDER & "/get_identities_repository.sh", 6, "", true)

' read the results
aFile   = PROGRAM_TMP & "/identities_p12"
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the listbox
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,oListbox.getItemCount())
      end if
```

```
wend
Close #iNumber

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/identities_p12")

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************


Sub closeChooseIdentityDialog

sIdentity = ""
oDialog.endExecute()

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************



Sub selectChooseIdentityDialog

oListBox = oDialog.GetControl("lstCertificates")
' get the selected P12
sIdentity = oListBox.getSelectedItem()
If sIdentity <>"" then
      oDialog.endExecute()
else
      MsgBox "Please select a P12 file!" & Chr(10) & Chr(10) & "To
close the window without selecting a P12 file press the Cancel
button."
end if

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************



Sub initVariables()
' initialize variables
PROGRAM_FOLDER="~/.securedoc"
PROGRAM_REPOSITORY=PROGRAM_FOLDER & "/repository"
PROGRAM_REPOSITORY_ENTITIES=PROGRAM_REPOSITORY & "/entities"
PROGRAM_REPOSITORY_IDENTITIES=PROGRAM_REPOSITORY & "/identities"
PROGRAM_TMP=PROGRAM_FOLDER & "/tmp"
```

```
End Sub
```

7.2 securedoc_crypt

```
Dim sFile
Dim sRecipients
Dim oDialog
Dim PROGRAM_FOLDER As String
Dim PROGRAM_REPOSITORY As String
Dim PROGRAM_REPOSITORY_ENTITIES As String
Dim PROGRAM_REPOSITORY_IDENTITIES As String
Dim PROGRAM_TMP As String

Sub Main

' initialize variables
PROGRAM_FOLDER="~/.securedoc"
PROGRAM_REPOSITORY=PROGRAM_FOLDER & "/repository"
PROGRAM_REPOSITORY_ENTITIES=PROGRAM_REPOSITORY & "/entities"
PROGRAM_REPOSITORY_IDENTITIES=PROGRAM_REPOSITORY & "/identities"
PROGRAM_TMP=PROGRAM_FOLDER & "/tmp"

sFile = ThisComponent.getLocation()
sRecipients = ""
BasicLibraries.LoadLibrary("Tools")
oDialog = LoadDialog("Standard", "dlgChooseRecipients")

' check whatever the document is saved and if it's name really exists
if sFile <> "" and FileExists(sFile) then
      ' preprocess the file name: remove the "file://" prefix
      sFile = ConvertFromURL(sFile)
      ' choose the recipients certificates
      getRecipientsCertificates()
      oDialog.Execute()
      ' ask for confirmation
      if sRecipients <> "" then
            if MsgBox("You are going to encrypt the file: " & sFile &
Chr(10) & Chr(10) & "for the following recipients: " & Chr(10) &
Chr(10) & sRecipients & Chr(10) & Chr(10) & "Are you sure?", 4) <> 6
then
                  exit sub
            else
                  ' write file name
                  aFile   = PROGRAM_TMP & "/crypt_input_file"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sFile
                  Close #iNumber
                  aFile   = PROGRAM_TMP & "/crypted_file_name"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sFile & ".p7m"
                  Close #iNumber
                  aFile   = PROGRAM_TMP & "/crypt_recipients"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sRecipients
```

```
                  Close #iNumber
                  ' run the external script
                  Shell (PROGRAM_FOLDER & "/crypt_file.sh", 6, "",
true)
                  ' read the script output
                  aFile   = PROGRAM_TMP & "/crypt_result"
                  iNumber = FreeFile()
                  Open aFile For Input Access Read As #iNumber
                  sResult = ""
                  While not eof(#iNumber)
                        Line Input #iNumber, sLine
                        sResult = sResult & sLine & Chr(13) & Chr(10)
                  wend
                  Close #iNumber
                  MsgBox sResult
            end if
      end if
else
      MsgBox "The file must be saved before you may crypt it!"
end if

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/crypt_input_file")
Kill(PROGRAM_TMP & "/crypted_file_name")
Kill(PROGRAM_TMP & "/crypt_recipients")
Kill(PROGRAM_TMP & "/crypt_recipients_certs")
Kill(PROGRAM_TMP & "/crypt_result")

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub getRecipientsCertificates

oListBox = oDialog.GetControl("lstCertificates")
oListBox.removeItems(0, oListBox.getItemCount())

' run the script to generate P12 list
Shell (PROGRAM_FOLDER & "/get_other_people_repository.sh", 6, "",
true)

' read the results
aFile   = PROGRAM_TMP & "/other_people_certificates"
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the listbox
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,0)
      end if
wend
Close #iNumber
```

```
' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/other_people_certificates")

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************



Sub closeChooseRecipientsDialog

sRecipients = ""
oDialog.endExecute()

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************



Sub selectChooseRecipientsDialog

oListBox = oDialog.GetControl("lstCertificates")
' get the selected certificates
sRecipients = oListBox.getSelectedItem()
If sRecipients <>"" then
      oDialog.endExecute()
else
      MsgBox "Please select at least one certificate!" & Chr(10) &
Chr(10) & "To close the window without selecting a certificate press
the Cancel button."
end if

End Sub
```

7.3 securedoc_repository

```
Dim oDialog
Dim page
Dim PROGRAM_FOLDER As String
Dim PROGRAM_REPOSITORY As String
Dim PROGRAM_REPOSITORY_ENTITIES As String
Dim PROGRAM_REPOSITORY_IDENTITIES As String
Dim PROGRAM_TMP As String
```

```
'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub Main

BasicLibraries.LoadLibrary("Tools")
oDialog = LoadDialog("Standard", "dlgRepositoryManagement")
oDialog.Execute()
page = 0

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub initVariables()
' initialize variables
PROGRAM_FOLDER="~/.securedoc"
PROGRAM_REPOSITORY=PROGRAM_FOLDER & "/repository"
PROGRAM_REPOSITORY_ENTITIES=PROGRAM_REPOSITORY & "/entities"
PROGRAM_REPOSITORY_IDENTITIES=PROGRAM_REPOSITORY & "/identities"
PROGRAM_TMP=PROGRAM_FOLDER & "/tmp"
End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub resetControlsValues()
oLabel = oDialog.getControl("lblCertificates")
oLabel.Text = "lblCertificates"
oButton = oDialog.getControl("btnPersonal")
oButton.getModel().SetPropertyValue("BackgroundColor",
255*256*256+255*256+255)
oButton.getModel().SetPropertyValue("TextColor", 0)
oButton = oDialog.getControl("btnOtherPeople")
oButton.getModel().SetPropertyValue("BackgroundColor",
255*256*256+255*256+255)
oButton.getModel().SetPropertyValue("TextColor", 0)
oButton = oDialog.getControl("btnTrustedIntermediateCA")
oButton.getModel().SetPropertyValue("BackgroundColor",
255*256*256+255*256+255)
oButton.getModel().SetPropertyValue("TextColor", 0)
oButton = oDialog.getControl("btnTrustedRootCA")
oButton.getModel().SetPropertyValue("BackgroundColor",
255*256*256+255*256+255)
oButton.getModel().SetPropertyValue("TextColor", 0)
oButton = oDialog.getControl("btnCertificateInfo")
oButton.Label = "Certificate Info"
```

```
oButton = oDialog.getControl("btnImport")
oButton.Label = "Import Certificate"
End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub getTrustedRootCACertificates

page = 1
initVariables()
resetControlsValues()
oLabel = oDialog.getControl("lblCertificates")
oLabel.Text = "Trusted Root CA Certificates List"
oButton = oDialog.getControl("btnTrustedRootCA")
oButton.getModel().SetPropertyValue("BackgroundColor", 255*256*256)
oButton.getModel().SetPropertyValue("TextColor",
255*256*256+255*256+255)
oListBox = oDialog.GetControl("lstCertificates")
oListBox.removeItems(0, oListBox.getItemCount())
oFile = oDialog.GetControl("fileImport")
oFile.Text = "Select a Trusted Root CA file to import"

' get trusted root ca list
Shell (PROGRAM_FOLDER & "/get_root_intermediate_repository.sh", 6,
"", true)

' read the results
aFile   = PROGRAM_TMP & "/trusted_root_ca"
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the listbox
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,oListbox.getItemCount())
      end if
wend
Close #iNumber

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/trusted_root_ca")
Kill(PROGRAM_TMP & "/trusted_intermediate_ca")

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************
```

```
Sub getTrustedIntermediateCACertificates

page = 2
initVariables()
resetControlsValues()
oLabel = oDialog.getControl("lblCertificates")
oLabel.Text = "Trusted Intermediate CA Certificates List"
oButton = oDialog.getControl("btnTrustedIntermediateCA")
oButton.getModel().SetPropertyValue("BackgroundColor", 255*256*256)
oButton.getModel().SetPropertyValue("TextColor",
255*256*256+255*256+255)
oListBox = oDialog.GetControl("lstCertificates")
oListBox.removeItems(0, oListBox.getItemCount())
oFile = oDialog.GetControl("fileImport")
oFile.Text = "Select a Trusted Intermediate CA file to import"

' get trusted root ca list
Shell (PROGRAM_FOLDER & "/get_root_intermediate_repository.sh", 6,
"", true)

' read the results
aFile   = PROGRAM_TMP & "/trusted_intermediate_ca"
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the listbox
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,oListbox.getItemCount())
      end if
wend
Close #iNumber

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/trusted_root_ca")
Kill(PROGRAM_TMP & "/trusted_intermediate_ca")

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub getOtherPeopleCertificates

page = 3
initVariables()
resetControlsValues()
oLabel = oDialog.getControl("lblCertificates")
oLabel.Text = "Other People Certificates"
oButton = oDialog.getControl("btnOtherPeople")
oButton.getModel().SetPropertyValue("BackgroundColor", 255*256*256)
oButton.getModel().SetPropertyValue("TextColor",
255*256*256+255*256+255)
oListBox = oDialog.GetControl("lstCertificates")
```

```
oListBox.removeItems(0, oListBox.getItemCount())
oFile = oDialog.GetControl("fileImport")
oFile.Text = "Select a User Certificate file to import"

' get entities
Shell (PROGRAM_FOLDER & "/get_other_people_repository.sh", 6, "",
true)

' read the results
aFile   = PROGRAM_TMP & "/other_people_certificates"
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the listbox
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,oListbox.getItemCount())
      end if
wend
Close #iNumber

' remove temporary files if they exists
on error resume next
'Kill(PROGRAM_TMP & "/other_people_certificates")

End Sub



'********************************************************************
'********************************************************************
'********************************************************************



Sub getPersonalCertificates

page = 4
initVariables()
resetControlsValues()
oLabel = oDialog.getControl("lblCertificates")
oLabel.Text = "Personal P12 identities"
oButton = oDialog.getControl("btnPersonal")
oButton.getModel().SetPropertyValue("BackgroundColor", 255*256*256)
oButton.getModel().SetPropertyValue("TextColor",
255*256*256+255*256+255)
oListBox = oDialog.GetControl("lstCertificates")
oListBox.removeItems(0, oListBox.getItemCount())
oButton = oDialog.getControl("btnCertificateInfo")
oButton.Label = "P12 Info"
oButton = oDialog.getControl("btnImport")
oButton.Label = "Import P12"
oFile = oDialog.GetControl("fileImport")
oFile.Text = "Select a P12 file to import as a profile"

' get entities
Shell (PROGRAM_FOLDER & "/get_identities_repository.sh", 6, "", true)

' read the results
aFile   = PROGRAM_TMP & "/identities_p12"
```

```
iNumber = FreeFile()
Open aFile For Input Access Read As #iNumber

' populate the list
While not eof(#iNumber)
      Line Input #iNumber, sLine
      If sLine <>"" then
            oListbox.additem(sLine,oListbox.getItemCount())
      end if
wend
Close #iNumber

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/identities_p12")

End Sub



'************************************************************************
'************************************************************************
'************************************************************************



Sub getSelectedCertificateDetails

initVariables()
oListBox = oDialog.GetControl("lstCertificates")
' get the selected certificatet
sSubject = oListBox.getSelectedItem()
If sSubject <>"" then
      ' write it's subject into a temporary file
      aFile   = PROGRAM_TMP & "/certificate_subject"
      iNumber = FreeFile()
      Open aFile for Output Access Write As #iNumber
      Print #iNumber, sSubject
      Close #iNumber

      ' if it is P12, ask for password
      if page = 4 then
            pass = ""
            pass = InputBox("Please enter the P12 password", "P12
Password", "")
            ' write it to the file
            aFile   = PROGRAM_TMP & "/password"
            iNumber = FreeFile()
            Open aFile for Output Access Write As #iNumber
            Print #iNumber, pass
            Close #iNumber
      end if


      ' run the script to obtain certificate details
      Shell (PROGRAM_FOLDER & "/get_certificate_info.sh", 6, page & "
" & sSubject, true)
      aFile   = PROGRAM_TMP & "/certificate_info"
      iNumber = FreeFile()
      Open aFile For Input Access Read As #iNumber
      ' read the file
```

```
	sCertInfo = ""
	While not eof(#iNumber)
		Line Input #iNumber, sLine
		sCertInfo = sCertInfo & sLine & Chr(13) & Chr(10)
	wend
	Close #iNumber
	' get validity info about the certificate
	aFile   = PROGRAM_TMP & "/certificate_info_validation"
	iNumber = FreeFile()
	Open aFile For Input Access Read As #iNumber
	' read the file
	sCertValidation = ""
	While not eof(#iNumber)
		Line Input #iNumber, sLine
		sCertValidation = sCertValidation & sLine & Chr(13) &
Chr(10)
	wend
	Close #iNumber

	' show the dialog window
	BasicLibraries.LoadLibrary("Tools")
	oDialog2 = LoadDialog("Standard", "dlgCertificateInfo")
	oText = oDialog2.getControl("txtCertificateInfo")
	oText.Text = sCertInfo
	oText = oDialog2.getControl("txtValidation")
	oText.Text = sCertValidation
	oDialog2.Execute()
else
	MsgBox "Please select a certificate from the list above!"
end if

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/certificate_subject")
Kill(PROGRAM_TMP & "/password")
Kill(PROGRAM_TMP & "/certificate_info")
Kill(PROGRAM_TMP & "/certificate_info_validation")

End Sub



'*********************************************************************
'*********************************************************************
'*********************************************************************



Sub deleteSelectedCertificate

initVariables()

if MsgBox("Are you sure you want to remove this Certificate/P12 from
repository ?", 4) <> 6 then
	exit sub
end if

oListBox = oDialog.GetControl("lstCertificates")
' get selected certificate
sSubject = oListBox.getSelectedItem()
If sSubject <>"" then
```

```
      ' write the subject into the file
      aFile   = PROGRAM_TMP & "/delete_certificate"
      iNumber = FreeFile()
      Open aFile for Output Access Write As #iNumber
      Print #iNumber, sSubject
      Close #iNumber

      ' get certificate info
      Shell (PROGRAM_FOLDER & "/delete_certificate.sh", 6, page & " "
& sSubject, true)
      aFile   = PROGRAM_TMP & "/delete_certificate_response"
      iNumber = FreeFile()
      Open aFile For Input Access Read As #iNumber
      ' read the file
      sResponse = ""
      While not eof(#iNumber)
            Line Input #iNumber, sLine
            sResponse = sResponse & sLine & Chr(13) & Chr(10)
      wend
      Close #iNumber

      ' show message
      MsgBox sResponse

      ' refresh repository list
      if page = 1 then getTrustedRootCACertificates
      if page = 2 then getTrustedIntermediateCACertificates
      if page = 3 then getOtherPeopleCertificates
      if page = 4 then getPersonalCertificates
else
      MsgBox "Please select a certificate from the list above!"
end if

' remove temporary files if they exists
on error resume next
Kill(PROGRAM_TMP & "/delete_certificate")
Kill(PROGRAM_TMP & "/delete_certificate_response")

End Sub



'**********************************************************************
'**********************************************************************
'**********************************************************************



Sub importCertificateP12

initVariables()
' check if any repository is selected
if page = 0 then
      MsgBox "Please select a repository by clicking one of the
buttons from the top of the window!"
      exit sub
end if
oFile = oDialog.GetControl("fileImport")
sFile = oFile.Text
' check if any certificate/p12 is selected
if sFile <> "" then
```

```
      ' check if file exists
      if FileExists(sFile) then
            ' check if it's size is greater than 0
            if FileLen(sFile) > 0 then
                  ' write it's name to the file
                  aFile   = PROGRAM_TMP & "/import_certificate_p12"
                  iNumber = FreeFile()
                  Open aFile for Output Access Write As #iNumber
                  Print #iNumber, sFile
                  Close #iNumber
                  ' run the import script
                  Shell (PROGRAM_FOLDER &
"/import_certificate_p12.sh", 6, page, true)
                  ' read the script output
                  aFile   = PROGRAM_TMP &
"/import_certificate_p12_result"
                  iNumber = FreeFile()
                  Open aFile For Input Access Read As #iNumber
                  sResult = ""
                  While not eof(#iNumber)
                        Line Input #iNumber, sLine
                        sResult = sResult & sLine & Chr(13) & Chr(10)
                  wend
                  Close #iNumber
                  MsgBox sResult

                  ' refresh repository list
                  if page = 1 then getTrustedRootCACertificates
                  if page = 2 then
getTrustedIntermediateCACertificates
                  if page = 3 then getOtherPeopleCertificates
                  if page = 4 then getPersonalCertificates
            else
                  MsgBox "ERROR: The file length is 0 or it is a
directory!"
            end if
      else
            MsgBox "ERROR: The file does not exists!"
      end if
else
      MsgBox "Please enter the Certificate/P12 file name you want to
import in the box from left!"
end if

' sterge fisierele temporare
on error resume next
Kill(PROGRAM_TMP & "/import_certificate_p12")
Kill(PROGRAM_TMP & "/import_certificate_p12_result")

End Sub
```