Chimaera 1.0 Tutorial & Reference

Jean-Patrick Giacometti

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Chimaera 1.0 Tutorial & Reference

Jean-Patrick Giacometti

N° 0193

Avril 1996

THÈME 1

Rapport technique

ISSN 0249-0803

Chimaera 1.0 Tutorial & Reference

Jean-Patrick Giacometti

Thème 1 — Réseaux et systèmes
Projet Rodeo

Rapport technique n˚ 0193 — Avril 1996 — 68 pages

Abstract: Chimaera is a model intended to secure data (messages) exchanges between applications. Chimaera uses the PEM (Privacy Enhanced Mail) format which enables the receiver to authentificate the sender identity using special informations (certificates, signature, hash-coding) held into the message. PEM is a model for e-mail authentification and certification. PEM complies to X509 standard as for the handling of certificates. Chimaera provides a way to generate PEM formatted mails. Such mails will be fully read by users running a similar PEM application, not necessarily Chimaera.

Basic Chimaera services are:
The certification of public keys of individuals through submission to the local certification authority.
The signing of e-mails under PEM format.

Key-words: PEM, electronic-mail, authentification, certification, X509.

(Résumé : tsvp)

TESI contract

Email :jpgsophia.inria.fr

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Chimaera 1.0
Manuel d'utilisation et de référence

Résumé : Chimaera est un modèle pour accroître la sécurité des échanges de messages. Il utilise le format PEM (Privacy Enhanced Mail) qui permet au destinataire de s'assurer de l'identité de l'expéditeur grâce aux champs spéciaux (certificats, signatures etc) contenus dans le message. PEM gère les certificats en respectant la norme X509. Chimaera permet d'envoyer et de recevoir des messages électroniques augmentés de ces informations spéciales. Ces messages sont lisibles par toute application utilisant PEM.

Chimaera offre deux services fondamentaux:
Les certifications de clés publiques par soumission à l'autorité de certification locale.
Les signatures des courriers électroniques au format PEM.

Mots-clé : PEM, courrier électronique, authentification, certification, X509.

Acknowledgements

Chimaera was developed at Inria within Christian Huitema's Rodeo project. Chimaera originated from Anas Altarah's PhD thesis ([1]), and was further developed by Pierre Pacchionni ([2]), Etienne Debroucker ([3], Anthony Charles ([3]), Christian Huitema, Aline Russeil ([4]) and J.-Patrick Giacometti.

Stelvio Verdiani and Sergio A. at Olivetti's Banking Security project also contributed to the issue.

1 Introduction

What is Chimaera? Chimaera is a model intended to secure data (messages) exchanges between applications. Chimaera uses the PEM (Privacy Enhanced Mail) format which enables the receiver to authentificate the sender identity using special informations (certificates, signature, hash-coding) held into the message. PEM is a model for e-mail authentification and certification. PEM complies to X509 standard as for the handling of certificates. Chimaera provides a way to generate PEM formatted mails. Such mails will be fully read by users running a similar PEM application, not necessarily Chimaera.

Basic Chimaera services are:

- *PK Certifications.* The certification of public keys of individuals thru submission to the local certification authority.
- *Signatures.* The signing of e-mails under PEM format.

2 Design

Chimaera is made of two basic commands, *himaera_ca* and *c*himaera_gs, implementing respectively the Certification Authority and the secret manager [in french, "Gestionnaire de Secret"]. No privilege is needed to run both, although chimaera_ca should be run only by the super user. Main role of *chimaera_ca* is to yield certificates to applications such as mailers. Such applications will run chimaera_gs. Thus an unique CA will suffice to serve a domain of related users. It is essential to note that the user that runs chimaera_ca is acting as a Certification Authority. When started *chimaera_gs* tries by default to IP Multicast a CA on the LAN. No knowledge of the host running the CA is necessary. In case the kernel does not implement Multicast, one has to cite the CA host, for instance by setting appropriately the *CHIMAERA_CAHOST* environment variable.

2.1 PEM Overview

PEM (Privacy Enhanced Mail) is a standard to exchange private data using e-mail standard. This PEM standard is described in RCF 1421,1422,1423 and 1424. The present implementation holds only asymmetric coding (RSA). A minimal Certificate Revocation List management is now supported on a local and off-line basis. A mail to be signed differs from an ordinary mail by the presence of the "Privacy:" header. On output this header is replaced by a PEM format header.

```
Example :

To: acharles\@pax.inria.fr
cc:
Subject: PEM...
Privacy: clear
-----
This is a PEM format mail.

End of Example.
```

The basic purpose of the "Privacy:" header is to tell whether the message is clear or crypted and in some way "how", i.e. which hash algorithm is used to ensure MIC.
Chimaera's PEM implementation supports:

- Privacy: clear (implicit default is md5)
- Privacy: clear, md5 (same as above)
- Privacy: clear, md2
- Privacy: encode (md5 is default)
- Privacy: encode, md5 (same as above)
- Privacy: encode, md2

Upon reception of a PEM formatted mail from a foreign originator, the certificates of both originator and issuer are inserted into the local base, namely the file *.chimaera.db*.

2.2 PEM message

The structure of a message PEM is the following.

- Certificate of the sender (signed by the local CA).
- Certificate of the CA (signed by itself).
- Text of the clear text message.
- Text of the message reduced by a hashing function (one way) then crypted by the RSA method with the private key of the sender. One calls the result of this processing "the signature of the message".

2.3 PEM Reception

Upon the reception of such a message, the recipient can verify easily the authenticity of the sender. It decrypts with the public key of the sender the signed text. It compares the result obtained with the text of the clear text message that it grinded previously in the same hashing function. The equality of these texts warranties thus the authenticity of the sender of the message.

3 Certification

3.1 Basic Certification Process

When a CA is run for the first time in a given context (repository), a message – known as a public key certification request – is sent to its supervising CA if any. In fact a mail is sent to the address found in the Chimaera configuration variable CA_FATHER; if none, the CA is considered to be a root CA – or PCA. When the CA's own local certificate expires, the certification procedure is run anew. This event generally occurs upon a GS request. The certification request is essentially asynchronous. No immediate answer form the upper certifying authority is waited for. Now one understands that a certifying CA may receive certification request in its mailbox. These request should be processed by running the command "chimaera_pca". The variable CHIMAERA_CA_BAL should hold the name of a directory to be used as repository input mailbox. As a consequence a PEM message carries more than the originator certificates. It holds a complete certification path up to the root certification authority.

3.2 Certificate

A certificate delivered to a user is a bunch of informations holding:

- The name specific of the user.
- The public key of this user.
- Some informations on the validity of this certificate.
- The name of the provider of this certificate (the local CA).
- The identification of the algorithm used to sign this certificate.

The whole signed by the CA.

3.2.1 Certificate Issuing

A unique user (normally *root*) uses Chimaera in mode "CA" . Namely as certification authority. It has for role to provide some certificates for the public keys of other users. All users trust this certification authority.

When a new CA is created, just after the generation of its public key / secret key pair and of its certificate signed by itself , a "request of certification of its public key" is sent to the superior CA in the shape of a e-mail. This superior CA to its electronic address in a environnement variable (CA_FATHER). Si this last one is missing, that means that there is no CA at the superior level and that it is in fact a PCA that is created.

This key certification request, generated automatically and in a systematic way for each creation of a new CA, is equally produced when the validity of the certificate of a CA comes to expiration. Nevertheless in this last case this is after a request of a GS that this event can take place.

It is important to note that this mecanism of key certification request is totally ASYNCHRONOUS. Namely that one does not wait an immediate answer of the superior CA. Which is absolutely normal because an OFFLINE document must normally equally reach to the superior CA organism in addition to this request electronic.

3.2.2 Certificate Issuing Request

The certification request specifies the "name of the subject" of the request and the "public key" in the shape of a self-signed certificate. This request holds two signatures, both computed with the "private key" of the requester.

- The signature on the self-signed certificate.
- The signature of a text any (by example: certification request)

First of all the composition of the self-signed certificate. These zones must be fed in the following way.

- Version: 0
- serial number of the certificate: 0
- Algo of signature: Algo with which this self-signed certificate is signed.
- issuer (provider): specified name of the requester
- Validity: not before: 700101120000Z not after: 700101120000Z
- Subjet: specified name of the requester.
- Infos about the public key of the subject:

The request by itself,example:

```
xxx     To: cert-service\@ca.domain
xxx     From: demandeur\@host.domain
xxx     --- BEGIN PREVACY-ENHANCED MESSAGE ---
xxx     Proc-Type: 4,MIC-ONLY
xxx     Content-Domain: RFC822
xxx     Originator-Certificate: < self-signed certificate of the
xxx                               requester>
xxx     MIC-info:RSA,RSA-MD2, < signature of the text of the
xxx                             requester >
xxx
xxx
xxx     < text of the requester >
xxx     --- END  PREVACY-ENHANCED MESSAGE ---
```

3.2.3 Certificate Issuing Reply

The requested CA signs the certificate only if the two signatures held in the request are both correct. The new certificate holds the "name of the subject" and the "public key" coming from the self-signed certificate. On the opposite the "serial number",the "name of the provider of the certificate", the "period of validity", "the algorithm of signature of this certificate" are to be chosen by the touched superior authority. The provider will be the superior CA next to the subject. It will sign this certificate with its private key and will transform the request in an answer holding this new certificate. In the certification answer one will add the path of certification up to the CPA. This path is normally in the local data base.

This answer is a PEM MIC-ONLY or MIC-CLEAR message. It has only a signature. The zone "MIC-Info" and the encapsulated text are taken directly from the certification request. The answer has the same type of processing (MIC-ONLY or MIC-CLEAR) that the request. This answer obtained one can register the new certificate in the data base of the requester and also the obtained certification path.

```
xxx     To: demandeur\@host.domain
xxx     From: cert-service\@ca.domain
xxx     --- BEGIN PREVACY-ENHANCED MESSAGE ---
xxx     Proc-Type: 4,MIC-ONLY
xxx     Content-Domain: RFC822
xxx     Originator-Certificate: < new certificate of the
xxx                              requester >
xxx     Issuer-Certificate: <  certificate of the provider of
xxx                            the certificate >
xxx     Issuer-Certificate: <  .................... >
xxx     Issuer-Certificate: <  .................... >
xxx     Issuer-Certificate: <  .................... >
xxx     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
xxx     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
xxx     MIC-info:RSA,RSA-MD2, < signature of the text of the
xxx                             requester >
xxx
xxx
xxx     < text of the requester >
xxx     --- END  PREVACY-ENHANCED MESSAGE ---
```

One sees here that one can send back many "Issuer-Certificate". That corresponds to a certification path. They are showed in order from the lowest level to the highest level.

4 CAs (The Certification Authorities Network)

4.1 Local Certification Authorities

One considers that there is a hierarchy among the CAs. Each CA can stock a certificate corresponding to its superior CA. A user can memorize the public keys, the certificates of all authorities located between the users and the root. That brings generally the user to know the public keys and the certificates of two or three authorities only. It needs to obtain some CPs (certification paths) only from the common point of trust (the CPA that is in this example V).

One sees thus here that it is interesting to have on each intermediate CA (in the hierarchy leading to the PCA) the list of the certificates of the superior CAs leading to the PCA.

Example:

```
                                                        V   *  fr  (PCA)
                                                           /
                                                          /
```

```
                  ( V«W» )      W   *  inria.fr
                                        /
                                       /
      ( V«W» , W «X» )    X    *  sophia.inria.fr
```

On sophia.inria.fr (X) it is interesting to have:

- The certificate of sophia.inria.fr signed by inria.fr
- The certificate of inria.fr signed by the PCA (fr)

On inria.fr (W) it is interesting to have

- The certificate of inria.fr signed by the PCA

So on each CA one builds easily the paths of certification up to the CA known by all, the PCA.

4.2 Primary Certification Authorities

Consider the processing of PEM mails received by a CA or a PCA: A CA can receive two types of PEM messages:

- Either some requests of key certification coming from inferior CAs (and of which it is thus the CA_FATHER).
- Or the answer to its last certification request near its CA superior (note that a PCA can not receive this kind of message).

A new programme, *chimaera_pca*, enabling to process these two types of messages must be run periodically by the users CAs and PCAs. It incorporates the new mails received in a given directory (its name is in the environment variable CHIMAERA_CA_BAL) to be able to study the ones after the others. The messages for which some anomalies are found (message that are not of PEM type , problem while in the control of the signature, etc...) are saved with the error code corresponding in their names of backup. One distinguish a "certification request of key" from an "answer to a certification request" by the fact that the first has no provider of certificate.

5 More about design

5.1 Certificate Cache

This note is to answer a question arised at Ivrea meeting: « is the RA the only interface to the cache? » whose corollary is: « what is to be ported to the PC? »

In today version of Chimaera the RA (Registration Authority) is embodied in the *chimaera_ca* command. The cache is incarnated by the RA data base file. The RA is connected to the network ans instantiated by running *chimaera_ca* with the option "-*k 1*" , which is the present syntax for the future option "-*no_key*". This suppose its cache may have been build off-line (on a secure sub net) either by a *chimaera_ca* + *chimaera_gs* process or by a *chimaster* process. Thus the CA/RA splitting is a bit virtual; the same component is running bit with or without the ability to deal with functions needing the knowledge of the authority's secret key. The RA/CA and the GS both interact in a similar way with their cache. To run *chimaera_ca* as a RA one faces the fact that, in this version, an authority needs its own keys and certificate to work. Thus one must copy the CA's cache at the beginning of the cache, get some keys, not the CA's, set the CA's environment (DName and UName) and run "*chimaera_ca* -*k 1*". So one has got a RA, running under the name of the CA, with different keys, serving certificates requests, and having no knowledge of the CA's key. On-line certification requests issued by GSs are discarded in the present version. This should be handled via PEM mails as is done for certification requests issued by CAs. The basic interface to the local cache (the cache of the user) is personified by the module chi_client.c. The remote mechanism is embodied into the module chi_server.c. This is also valid for the CA database cache.

As long as an application does not need the secret key of the user (resp. of the CA), it can access the local cache (resp. the CA database). When a processing involves the user's (resp. CA's) secret, the chi_client.c interface yields the mechanism to access the local server, i.e. the GS (resp. the CA).

The GS needs to connect to a CA only at first run or on expiration of the certificates. If it fails to, the user must try another way to get a valid certificate for himself. One possible way to get it is to run a *chimaera_ca* command in an environment with no local cache, with a CA_FATHER set to the real CA administrator e-mail, with a proper mbox setting and finally with the distinguished name set to himself. We may have for instance:

```
django:...$ env|fgrep CHI
CHIMAERA\_CA\_BAL=/chimaera/w-ra/PCA\_BAL
CHIMAERA\_CA\_FATHER=BigBoss\@sophia.inria.fr
CHIMAERA\_DNAME=<C=FR,O=Inria,OU=ROCQUENCOURT,CN=jpg>
django:...$
```

This will cause a certification request PEM mail to be sent. If an answer is replied, the user can integrate its new certificate by running a chimaera_pca command. Now he has a local cache holding two entries:

```
<himself>BigBoss
<BigBoss>BigBoss
```

that is a certificate for himself signed by the authority followed of a certificate of authority by itself. With an ordinary text editor the user should permute the two certificates in order to have the CA's certificate in first position, because the GS assumes that the first certificate of the cache is the one of the default CA. The GS can be run on a local cache provided by off-line means, with the condition that the first certificate be the one of the CA. The local file holding the keys of the user may have been generated either locally by the user itself running the GS in presence of a RA or off-line by an authority running *chimaster*; the latter implies the passing of the *.chimaera.key* file with the login password to the user. In the present version of Chimaera, with no such key file, the GS emits a broadcast request to get the CA certificate instead of looking for it in the local cache.

So we can asnwer to the first question: « The RA is the only interface to the secret. The cache is public information.»

An answer to the second question is to have a minimal port located at the key building step and at the signing step.

5.2 Certification Path

A certification path is in fact a chain of points of trust that allows a user of obtain the public key of another user. One of the services of Chimaera consists to put in network the CAs in order to set " some certification paths " towards a CA known by most people : the PCA.

Example:

```
                      Fictitious disposition of the CAs

  legend:  W«X» : Certificate of X issued by the certification
                      authority W.
             Xp     : public key of X
```

The certification path of C towards B (written C–>B) enabling C of obtain the public key of B is the following:

```
  Bp  =  Xp . X«W» . W«V» . V«Y» . Y«Z» . Z«B»
```

5.2.1 Certification Path Issuing

This is about the principle of creation and management of the certifications paths.

When a new CA will be created, after having generated its public key / private key pair, will send a request of certification of its public key to the superior CA in the hierarchy. This service is provided for in the PEM protocole RFC 1421, we will itemize it further. In answer to this request, the superior CA will send back the certificate of the public key of the requester signed by itself. By the way in its answer it will add the list of the certificates that it owns and corresponding to the path of certification of this one up to its PCA. the CA then should insert these received informations in its local data base . One must thus provide in Chimaera for a table holding the list of the acknowledged by all as valid PCAs in to way to know until where one must go back in the searches of certificates. Step by step, upon this public key certification request procedure, the lowest CAs in the tree, created thus normally after theirs father CAs, will collect the certifications paths of these up to the PCA .

5.2.2 CP Issuing Mecanism

The request by a CA of the certification of its public key by a superior CA is described in RFC 1421 PEM. This service of key certification, signs a certificate holding a "specified subject name" and a "public key".

- It takes the certification request
- Signs the certificate built from the request
- Send back an answer of certification of answer holding the new certificate.

More precisely, Treatment of a "certification request of key":

- Extraction of the electronic address of the sender to send the answer.
- Verification of the signature of the self-signed certificate.
- Verification of the signature of the text.
- Interactive request on the screen if one must sign this certificate.
- If yes request of the capture of the password
- Generation of the new certificate for the requester holding: "the name of the requester" and its "public key" coming from the self-signed certificate, the new attributed "serial number", a "period of validity" equal to the one of the CA processing the request to which one a few days (ADD_VALIDITY constant), and the name of the CA processing the request as "name of the provider of certificate". One signs then this new certificate with the private key of the CA.

This work done, the new certificate is send back to the requester in the shape of an e-mail. In this answer one adds the certification path up to the PCA. This path is in the local data base. This answer is a message PEM MIC-ONLY. It has only one signature. The encapsulated text is taken directly from the request.

```
xxx     To: demandeur\@host.domain
xxx     From: cert-service\@ca.domain
xxx     --- BEGIN PREVACY-ENHANCED MESSAGE ---
xxx     Proc-Type: 4,MIC-ONLY
xxx     Content-Domain: RFC822
xxx     Originator-Certificate: < new certificate of the
xxx                               requester >
xxx     Issuer-Certificate: <  certificate of the provider of
                                 the certificate >
xxx     Issuer-Certificate: <  .................... >
xxx     Issuer-Certificate: <  .................... >
xxx     Issuer-Certificate: <  .................... >
xxx     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
xxx     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
xxx     MIC-info:RSA,RSA-MD2, < signature of the text of the
xxx                             requester >
xxx
xxx
xxx     < text of the requester >
xxx     --- END  PREVACY-ENHANCED MESSAGE ---
```

One sees here that one can send back many "Issuer-Certificate". This corresponds to a certification path. They are showed in the order from the lowest level to the highest level.

5.2.3 CP Feeding Mecanism

Processing of an "answer to a key certification":

- Verification of the signature of the new certificate (originator certificate)
- Verification of the signatures of the certificates constituting the path of certification (issuers certificates).
- Verification of the equality between the public key of the CA (in the data base) and the one contained in the answer
- Deposit of the new certificate in the data base.
- Deposit of the certificates of the issuers in the data base.

5.2.4 Certification Process Errors

```
======================================================================
                 Error handling
======================================================================
 The messages for which anomalies are found
are memorized with the error code in their names of backup.

A name of message backup has the following form:
         .(date)-(hour).(number in directory).err(Error Code)
```

```
Example: .940317-191237.1.err-2

=====================================================================
 Code                   Spelling of the error
 error
=====================================================================
-1  Problem upon the open of the message to process
-2  The message is not pas a message PEM
-3  Problem upon the open of the message to process (before parse)
-4  Problem upon the call of the function read_ca_algo
-5  Problem upon the call of the function read_ca_name
-6  Problem upon the open of the message to process

    =================================================================
              TRAITEMENT D'UNE DEMANDE DE CERTIFICATION DE CLEF
              Processing of a Key certification request

-7  Signature control of the self-signed certificate fails
-8  Problem on return of the call of the function pem_canonicalize
-9  Signature control of the text fails
-10 Problem on return of the call of the function pem_decanonicalize
-23 If negative answer to the interactive request of certification
-11 Problem upon the attribution of a serial number
-12 Read certificate of the CA in data base fails
-13 Problem on return from the call of the function asn1_time
-14 Problem while in the calculus of the validity of the certificate
-15 Typed-in password is invalid
-16 Read of the key fails upon the call read_rsakey
-18 Problem upon the sending of the answer to the requester

   =================================================================
              TRAITEMENT DE LA REPONSE A LA DEMANDE DE CERTIFICATION
              Processing of the answer to a certification request

-24 Signature control of the new certificate fails
```

```
-25 Signature control of the certificate of an issuer fails
-19 Read certificate of the CA in data base fails
-20 Public key of the answer differs from the one of the CA
-21 Error while in the update of certificate of the CA in the base
-22 Error while in the update certificate of a issuer in the base
====================================================================
```

5.3 Secure Certificate

Rules are twice:

1. we trust what we signed ourselves, with a degree of confidence – the certification mode.
2. we reject signature by an hierarchy-inferior issuer unless rule 1 applies.

The certificate verification procedure is two steps

1. Check that this alleged certificate for A signed by B is not a bogus, first by checking the signature. Thus we need to get a certificate for the issuer B. If we can't get one, return error. Now we have a certificate for B. Check signature. if no match, reject.
2. Look for a certificate for that user A in the local data base. If one is found Check whether the certificate for A from our data base has been issued (signed) by ourselves.

```
         If so, and if the keys match, and
                if A < Us, then ok,
                else    if issued as PCA trustable, ok
                        else reject.
step 2
        else - not signed by us
                If A >= B reject.
                else get\_one\_valid\_best\_certificate\_for(B)
                     among the following preferences:
                        myself
                        any issuer > subject B
                        others

                if can’t get <B>[X] reject <A>[B].
                else time to get <B>[Y]
                        we verify the signature of <A>[B].
                        if ok recurse back from step 2 for issuer
                           of B
                        else reject.
```

6 CRLs (certificate revocation lists)

6.1 CRL Management

Fundamentals of CRL management are:

1. About the command *chimaster*

   chimaster uses the current time as revocation date. It calls ‘crl_revoke_certificate()’ to revoke a certificate thru the line command ‘d <user-name>’.

2. How to create the crls cache?

   Use the command *chimaster* as the CA. The init of the implicit crl store file is triggered by the first ‘d someone’ following a ‘u someone’.

3. How to test the functionality?

   Run two differents CA in, say, directories ca and ca-tmp. Initialize some users at both. Make some revocations. Bow the two CAs exchange their crls: run *chimaster,* sign with ’g’ send with ‘s <CA-email>’; run *chimaera_pca* on both dirs (ca and ca-tmp) Note: Here, only the own crl of the ca is sent.

4. About the revocation list cache implementation

   A crl has a limited size because an expired certificate is a fortiori de facto revoked. So keep only last Update from each issuer.

5. How to build and send a storage request

   - First get a certificate issued by us and revoke it. (*chimaster* by hand) answer ‘d’ of *chimaster* -> user_revoke...
   - Second emit a storage request to father. (chimaster by hand thru the line command ‘s father’).

6. How to emit a retrieval request?

   In *chimaster*, just say ‘1 <dname>’ then ‘i’.

7. How to list the contents of the crls store?

   This is provided by the line command ‘L’ of *chimaster*. Mind that ‘’l’ lists the certificate cache. The entries from issuer <dname> can be selectively listed with the line command ‘i’, once the issuer’s X500-name register has been set by the line command ‘1 <dname>’.

8. Note: the compilation option ON_LINE of chi_trtdmdf.c (future chi_ctrust.c) enables to have a rpc exchange with a gs to put a certificate instead of a CA-like local write to ./.chimaera.db

```
|========================|
| Summary of the section |
|========================|
```

Local handling of CRL is done with the command ‘*chimaster*’ run by the CA. Incoming PEM CRL requests and replies are dealed with by the mail command ‘*chimaera_pca*’.

6.2 CRL Implementation

Some basic functions are presented to give a feeling of the matter. As to the API, it is extended with the function check_certificate_legality.

```
/*---------------------------------------*
 * Function Name : check_certificate_legality              *
 * Description   : check if the current time of use is not  *
 *                 before and not after the validity and if *
 *                 it has not yet been revoked              *
 * Mode          : Uses a Revocation  store                 *
 * Arguments     : certificate to check                     *
 * Return        : 0 if ok, 1 else                          *
 *---------------------------------------*/
int check_certificate_legality (to_check)
  Certificate *to_check;
```

The basic functions for storage interface are: store_revocation_list, crl_update, read_revocation_list, is_certificate_usable, crl_revoke_certificate.

```
 * =====================
 * chi_crl_request_store
 * =====================

 * returns :  1 if success, -1 otherwise

int chi_crl_store_request(issuer, recipient, crlfile)
certificates * issuer;
char * recipient;
char * crlfile;
```

‘chi_crl_request_store()’ is called once by the command ‘chimaster’, thru the line command ‘s <recipient>’.

```
int chi_reply_crl_retrieval(issuer , recipient, crlfile)
    ---------------
certificates *issuer;
char * recipient ;
char * crlfile ;


 Argument      : - issuer’s certificate
                 - recipient name
                 - crl store file
 Return        : 1 if success, -1 otherwise

-----
```

‘chi_reply_crl_retrieval()’ is called by ‘chi_trtdmdf()’ which is itself called till now only by the command ‘chimaera_pca’.

```
 * ======================
 * crl_revoke_certificate
 * ======================

 * -1- check is certificate is indeed usable
 * -2- if an entry already exists, check dates, keep sooner
 * -3- else enter. Here we may have to create the crl.
 * returns 0 on success, -1 on failure

int crl_revoke_certificate(filename, certificate, fromDate, mirror)
char *        filename;
Certificate * certificate;
asn1_field *  fromDate;
int mirror;
```

‘crl_revoke_certificate()’ is called once by the command chimaster.

```
 * =====================
 * store_revocation_list
 * =====================
 * if there is already a revocation list for this issuer, check dates,
 * keep last.

 Function Name : store_revocation_list
 Description   : apply revocation notice
 Arguments     : filename          - the file who contain the crls
                revocation_list   - the notice of revocations
                mirror            - back file or not....
 Return        : 1 if success, -1 otherwise if error.
```

‘store_revocation_list()’ is called twice, once by ‘chi_trtdmdf()’ and once by ‘crl_update()’.

```
 * ==========
 * crl_update
 * ==========

 * sign the CA crl
 * -1- get my crl and time
 * -2- update lastUpdate field
 * -3- set my algo
 * -4- sign the object
 * -5- store
 * returns : 1 on success, -1 if not.
```

'crl_update()' is called once by the command 'chimaster'. Purpose is to provide a CA-crl sign. To determine when to do so is from the CA's responsability. It is done with the chimaster command by the line command 'g'.

```
|========================================|
| Summary of the section Implementation |
|========================================|
```

The present implementation involves the following modules:

chimaster.c

chi_trtdmdf.c

chi_crls.c

chi_stor.c

chi_repr.c

chi_retr.c

6.3 CRL Management Specifications

The following are mainly taken form IAB's RFCs 1421, 1422, 1423, 1424.

The present release provides basic CRLs management by CAs and an API function to check the legality of a certificate.

6.3.1 Revocation Lists

The RA servers requests for certificate revocation lists from a local cache. It stores requests for certificate revocation for off line processing by the CA. [chi #1] The Chimaera cache is by default the file ./.chimaera.crl. There is an interface command for specifying which certificates should be revoked. [chi #2] This interface command is called ‘chimaster’ in Chimaera. [chi #3] Chimaera enables to revoke a user instead of a certificate. [chi #4] The line command to revoke a user is ‘d <unix user name>’. [chi #5] The chimaster command is run off-line by the CA manager. The syntax of the revocation lists shall follow the X.500 92/93 standards.

6.3.2 CRL Management paradigm

X.500 makes provision for the storage of CRLs as directory attributes associated with CA entries. Thus, when X.500 directories become widely available, UAs can retrieve CRLs from directories as required. In the interim, the IPRA will coordinate with PCAs to provide a robust database facility which will contain CRLs issued by the IPRA, by PCAs, and by all CAs.

Access to this database will be provided through mailboxes maintained by each PCA.

Every PEM UA must provide a facility for requesting CRLs from this database using the mechanisms defined in RFC 1424. [chi #6] This facility is the line command ‘i’ of *chimaster*.

Thus the UA must include a configuration parameter which specifies one or more mailbox addresses from which CRLs may be retrieved.

Access to the CRL database may be automated, e.g., as part of the certificate validation process (see Section 3.6) or may be user directed. [chi #7] The user directed form in Chimaera is the command ‘*chimaster*’.

Responses to CRL requests will employ the PEM header format specified in RFC 1421 for CRL propagation.

As noted in RFC 1421, every PEM UA must be capable of processing CRLs distributed via such messages. [chi #8] This is done by hand in Chimaera with the command ‘*chimaera_pca*’.

This message format also may be employed to support a "push" (versus a "pull") model of CRL distribution, i.e., to support unsolicited distribution of CRLs.

CRLs received by a PEM UA must be validated (A CRL is validated in much the same manner as a certificate, i.e., the CIC (see RFC 1113) is calculated and compared against the decrypted signature value obtained from the CRL. See Section 3.6 for additional details related to validation of certificates.) prior to being processed against any cached certificate information.

Any cache entries which match CRL entries should be marked as revoked, but it is not necessary to delete cache entries marked as revoked nor to delete subordinate entries.

In processing a CRL against the cache it is important to recall that certificate serial numbers are unique only for each issuer and that multiple, distinct CRLs may be issued under the same CA DN (signed using different private components), so care must be exercised in effecting this cache search. (This situation may arise either because an organizational CA is certified by multiple PCAs, or because multiple residential CAs are certified under different PCAs.)

This procedure applies to cache entries associated with PCAs and CAs, as well as user entries.

The UA also must retain each CRL to screen incoming messages to detect use of revoked certificates carried in PEM message headers.

Thus a UA must be capable of processing and retaining CRLs issued by the IPRA (which will list revoked PCA certificates), by any PCA (which will list revoked CA certificate issued by that PCA), and by any CA (which will list revoked user or subordinate CA certificates issued by that CA).

Among the procedures articulated by each PCA in its policy statement are procedures for the maintenance and distribution of CRLs by the PCA itself and by its subordinate CAs. The frequency of issue of CRLs may vary according to PCA-specific policy, but every PCA and CA must issue a CRL upon inception to provide a basis for uniform certificate validation procedures throughout the Internet hierarchy. The IPRA will maintain a CRL for all the PCAs it certifies and this CRL will be updated monthly. Each PCA will maintain a CRL for all of the CAs which it certifies and these CRLs will be updated in accordance with each PCA's policy. The format for these CRLs is that specified in Section 3.5.2 of the document.

In the absence of ubiquitous X.500 directory services, the IPRA will require each PCA to provide, for its users, robust database access to CRLs for the Internet hierarchy, i.e., the IPRA CRL, PCA CRLs, and CRLs from all CAs. The means by which this database is implemented is to be coordinated between the IPRA and PCAs. This database will be accessible via email as specified in RFC 1424, both for retrieval of (current) CRLs by any user, and for submission of new CRLs by CAs, PCAs and the IPRA.

Individual PCAs also may elect to maintain CRL archives for their CAs, but this is not required by this policy.

6.3.3 How to keep lists reasonably up to date

For CRL management, each PCA must specify the frequency with which it will issue scheduled CRLs.

It also must specify any constraints it imposes on the frequency of scheduled issue of CRLs by the CAs it certifies, and by subordinate CAs. Both maximum and minimum constraints should be specified.

Since the IPRA policy calls for each CRL issued by a CA to be forwarded to the cognizant PCA, each PCA must specify a mailbox address to which CRLs are to be transmitted.

The PCA also must specify a mailbox address for CRL queries.

If the PCA offers any additional CRL management services, e.g., archiving of old CRLs, then procedures for invoking these services must be specified.

6.3.4 CA Responsibilities for CRL Management

As X.500 directory servers become available, CRLs should be maintained and accessed via these servers. However, prior to widespread deployment of X.500 directories, this document adopts some additional requirements for CRL management by CAs and PCAs. As per X.509, each CA is required to maintain a CRL (in the format specified by this document in Appendix A) which contains entries for all certificates issued and later revoked by the CA.

Once a certificate is entered on a CRL it remains there until the validity interval expires.

The interval at which a CA issues a CRL is not fixed by this document, but the PCAs may establish minimum and maximum intervals for such issuance.

As noted earlier, each PCA will provide access to a database containing CRLs issued by the IPRA, PCAs, and all CAs.

In support of this requirement, each CA must supply its current CRL to its PCA in a fashion consistent with CRL issuance rules imposed by the PCA and with the next scheduled issue date specified by the CA (see Section 3.5.1).

CAs may distribute CRLs to subordinate UAs using the CRL processing type available in PEM messages (see RFC 1421). [chi #9] This is done by the line command '*g*' of the command '*chimaster*'.

CAs also may provide access to CRLs via the database mechanism described in RFC 1424 and alluded to immediately above.

7 X.509 CRL Specifications

X.509 states that it is a CA's responsibility to maintain: "a time- stamped list of the certificates it issued which have been revoked." There are two primary reasons for a CA to revoke a certificate, i.e., suspected compromise of a private component (invalidating the corresponding public component) or change of user affiliation (invalidating the DN). The use of Certificate Revocation Lists (CRLs) as defined in X.509 is one means of propagating information relative to certificate revocation, though it is not a perfect mechanism. In particular, an X.509 CRL indicates only the age of the information contained in it; it does not provide any basis for determining if the list is the most current CRL available from a given CA.

The proposed architecture establishes a format for a CRL in which not only the date of issue, but also the next scheduled date of issue is specified.

Adopting this convention, when the next scheduled issue date arrives a CA (Throughout this section, when the term "CA" is employed, it should be interpreted broadly, to include the IPRA and PCAs as well as organizational, residential, and PERSONA CAs.) will issue a new CRL, even if there are no changes in the list of entries.

In this fashion each CA can independently establish and advertise the frequency with which CRLs are issued by that CA.

Note that this does not preclude CRL issuance on a more frequent basis, e.g., in case of some emergency, but no system-wide mechanisms are architected for alerting users that such an unscheduled issuance has taken place.

This scheduled CRL issuance convention allows users (UAs) to determine whether a given CRL is "out of date," a facility not available from the (1988) X.509 CRL format.

The description of CRL management in the text and the format for CRLs specified in X.509 (1988) are inconsistent. For example, the latter associates an issuer distinguished name with each revoked certificate even though the text states that a CRL contains entries for only a single issuer (which is separately specified in the CRL format). The CRL format adopted for PEM is a (simplified) format consistent with the text of X.509, but not identical to the accompanying format.

7.1 PEM CRL Format

Appendix A contains the ASN.1 description of CRLs specified by this document. This section provides an informal description of CRL components analogous to that provided for certificates in Section 3.3.

1. signature (signature algorithm ID and parameters)
2. issuer
3. last update
4. next update
5. revoked certificates

The "signature" is a data item completely analogous to the signature data item in a certificate. Similarly, the "issuer" is the DN of the CA which signed the CRL. The "last update" and "next update" fields contain time and date values (UTCT format) which specify, respectively, when this CRL was issued and when the next CRL is scheduled to be issued. Finally, "revoked certificates" is a sequence of ordered pairs, in which the first element is the serial number of the revoked certificate and the second element is the time and date of the revocation for that certificate.

7.2 Certificate Validation

If a message arrives from an originator whose public component is held in the recipient's cache (and if the cache is maintained in a fashion that ensures timely incorporation of received CRLs), the recipient can immediately employ that public component without the need for the certificate validation process described here. (For some digital signature algorithms, the processing required for certificate validation is considerably faster than that involved in signing a certificate. Use of such algorithms serves to minimize the computational burden on UAs.

7.3 Validation details

Here are some Validation Procedure details.

Each certificate also must be checked against the current CRL from the certificate's issuer to ensure that revoked certificates are not employed.

If the UA does not have access to the current CRL for any certificate in the path, the user must be warned.

Again, the form of the warning is a local matter. For example, the warning might indicate whether the CRL is unavailable or, if available but not current, the CRL issue date should be displayed.

Local policy may prohibit use of a public component which cannot be checked against a current CRL, and in such cases the user should receive the same information provided by the warning indications described above.

If any revoked certificates are encountered in the construction of a certification path, the user must be warned. The form of the warning is a local matter, but it is recommended that this warning be more stringent than those previously alluded to above.

For example, this warning might display the issuer and subject DNs from the revoked certificate and the date of revocation, and then require the user to provide a positive response before the submission or delivery process may proceed.

In the case of message submission, the warning might display the identity of the recipient affected by this validation failure and the user might be provided with the option to specify that this recipient be dropped from recipient list processing without affecting PEM processing for the remaining recipients.

Local policy may prohibit PEM processing if a revoked certificate is encountered in the course of constructing a certification path.

When non human interaction is involved, a compliant PEM implementation must provide parameters to enable a process to specify whether certificate validation will succeed or fail if any of the conditions arise which would result in warnings to a human user.

7.4 CRL Storage

The CRL storage service stores CRLs. The service takes a CRL-storage request (see Section 3.3) specifying the CRLs to be stored, stores the CRLs, and returns a CRL-storage reply (see Section 3.4) acknowledging the request.

The certification authority stores a CRL only if its signature and certification path are valid, following concepts in RFC 1422 (Although a certification path is not required in a CRL-storage request, it may help the certification authority validate the CRL.)

7.5 CRL Retrieval

The CRL retrieval service retrieves the latest CRLs of specified certificate issuers. The service takes a CRL-retrieval request (see Section 3.5), retrieves the latest CRLs the request specifies, and returns a CRL-retrieval reply (see Section 3.6) containing the CRLs.

There may be more than one "latest" CRL for a given issuer, if that issuer has more than one public key (see RFC 1422 for details).

The CRL-retrieval reply includes a certification path from each retrieved CRL to the RFC 1422 Internet certification authority. It may also include other certificates such as cross-certificates that the certification authority considers helpful to the requestor.

7.6 CRL-storage request

A CRL-storage request is an RFC 1421 CRL-type privacy-enhanced message containing the CRLs to be stored and optionally their certification paths to the RFC 1422 Internet certification authority.

Example:

```
To: cert-service@ca.domain
From: requestor@host.domain

---BEGIN PRIVACY-ENHANCED MESSAGE---
Proc-Type: 4,CRL
CRL: <CRL to be stored>
Originator-Certificate: <CRL issuer’s certificate>
CRL: <another CRL to be stored>
Originator-Certificate: <other CRL issuer’s certificate>
---END PRIVACY-ENHANCED MESSAGE---
```

7.7 CRL-storage reply

A CRL-storage reply is an ordinary message acknowledging the storage of CRLs. No particular syntax is specified.

7.8 CRL-retrieval request

A CRL-retrieval request is a new type of privacy-enhanced message, distinguished from RFC 1421 privacy-enhanced messages by the process type CRL-RETRIEVAL--REQUEST.

The request has two or more encapsulated header fields: the required "Proc-Type:" field and one or more "Issuer:" fields. The fields must appear in the order just described. There is no encapsulated text, so there is no blank line separating the fields from encapsulated text.

Each "Issuer:" field specifies an issuer whose latest CRL is to be retrieved. The field contains a value of type Name specifying the issuer's distinguished name. The value is encoded as in an RFC 1421 "Originator-ID-Asymmetric:" field (i.e., according to the Basic Encoding Rules, then in ASCII). Example:

```
To: cert-service\@ca.domain
From: requestor\@host.domain

---BEGIN PRIVACY-ENHANCED MESSAGE---
Proc-Type: 4,CRL-RETRIEVAL-REQUEST
Issuer: <issuer whose latest CRL is to be retrieved>
Issuer: <another issuer whose latest CRL is to be retrieved>
---END PRIVACY-ENHANCED MESSAGE---
```

An example Recipient-ID field for the asymmetric case is as follows:

```
Recipient-ID-Asymmetric:
 MFExCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5j
 LjEPMA0GA1UECxMGQmV0YSAxMQ8wDQYDVQQLEwZOT1RBUlk=,66
```

This example field includes the printably encoded BER representation of a certificate's issuer distinguished name, along with the certificate serial number 66 as assigned by that issuer.

7.8.1 Issuing Authority Subfield

For the asymmetric key management case, the IA identifier subfield will be formed from the ASN.1 BER representation of the distinguished name of the issuing organization or organizational unit. The distinguished encoding rules specified in Clause 8.7 of Recommendation X.509 ("X.509 DER") are to be employed in generating this representation. The encoded binary result will be represented for inclusion in a transmitted header using the procedure defined in Section 4.3.2.4 of this RFC.

7.8.2 Version/Expiration Subfield

For the asymmetric key management case, the version/expiration subfield's value is the hexadecimal serial number of the certificate being used in conjunction with the originator or recipient specified in the "Originator-ID-Asymmetric:" or "Recipient-ID-Asymmetric:" field in which the subfield occurs.

7.9 CRL-retrieval reply

A CRL-retrieval reply is an RFC 1421 CRL-type privacy-enhanced message containing retrieved CRLs, their certification paths to the RFC 1422 Internet certification authority, and possibly other certificates.

Since the reply is an ordinary privacy-enhanced message, the retrieved CRLs can be inserted into the requestor's database during normal privacy-enhanced mail processing. The requestor can forward the reply to other requestors to disseminate the CRLs.

Example:

```
To: requestor\@host.domain
From: cert-service\@ca.domain

---BEGIN PRIVACY-ENHANCED MESSAGE---
Proc-Type: 4,CRL
CRL: <issuer's latest CRL>
Originator-Certificate: <issuer's certificate>
CRL: <other issuer's latest CRL>
Originator-Certificate: <other issuer's certificate>
```

```
   ---END PRIVACY-ENHANCED MESSAGE---
```

[chi #10] There may be more than one revocation for a certificate, each with a different revocation date.

—

8 Release Notes

This release (Date: Apr 19 1995) includes Certificate Revocation List basic support. Two new test programs – chi_tapi and chi_tcrypt – are added. The release is made of 8 directories, namely asn1, bin, demo, doc, etc, lib mailer, man, src.

8.1 bin (The commands)

- *chimaera_gs* (the secret manager)
- *chimaera_ca* (the certification authority)
- *chimaera_pca* (the off-line certificator)
- *chimaster* (the off-line ca)
- *pem_create* (input PEM processing: sign)
- *pem_show* (output PEM processing: check).

8.2 demo (data for demonstrations)

```
README for the Chimaera Demo
-------------------

Contents
-----
1 - Setting the demo environment
2 - Testing the API
2.1 - Running "CA -d"
2.2 - Running "GS -d"
```

1 - Setting the demo environment

The main concern of Chimaera is security. Thus you should find a bit hard to run a demo without getting some password knowledge. We will suppose you know the ad hoc password.
If this is not the case and you want to test the API, you should find or build a test data base. This can be made with Chimaera by mimicking the demo set.
You must take care of some constants in file chi_tcli.c or adapt.

From this, first set in your environment the CHIMAERA_HOME variable to the root of the chimaera distribution.
The commands (scripts) from here can be found in the directory etc. So, assuming you run /bin/sh, you should set in your env the PATH var:

```
PATH=$CHIMAERA_HOME/etc:$PATH
```

2 - Testing the API

```
Now you may test the API.
Briefly:

0- export CHIMAERA_HOME=somewhere
   export PATH=$CHIMAERA_HOME/etc:$PATH

1- A_host> CA  -d

2- B_host> GS  -d
3- B_host> API -d

2.1 - Running "CA -d"
--------------
  Get on some host.
If you wish to see test the building of the key, simply
remove the files .chimaera.key and .chimaera.db in directory
$CHIMAERA_HOME/demo/w-ca.

2.2 - Running "GS -d"
--------------
 Get on another host.
This is more realistic and in practice
it avoids to explicitely name the port to use. Default port
is used. Of course, this is not compulsory but in the case
you would run CA and GS on the same machine you will have to
modify the scripts accordingly.

2.3 - Running "API -d"
---------------
  Stay on the same host as GS.
This is compulsory; this constraint is a design feature of
Chimaera’s architecture.
You will be prompted for sign several time; type in the
password at the cursor then confirm by a click on the yes
button.
```

```
3 - Testing the off-line Certification procedure
--------------------------------
  Now you may test off-line certification.
Briefly: in one place
1- > CA -de
3- > PCA -dp
and in another place
2- > RA -de
4- > PCA -dr

3.1 - Running "CA -de"
--------------
  Purpose is to build the base of <C=FR,O=Inria>.
You become a potential certifier.

3.2 - Editing the Certification profile
--------------------------
  Concern is about thefile demo/w-ra/.profile.ra
Purpose is to set the variable CHIMAERA_CA_FATHER to the
uid e-mail of the physical person who run CA just above
the certifier).

3.3 - Running "RA -de"
---------------
  Purpose is to build the base for <C=FR,O=Inria,
U=ROCQUENCOURT>. The demo is prettier if you have
impersonated another user than the above certifier.
A PEM mail request for certification will be sent to
the certifier.

3.4 - Running "PCA -dp"
---------------
  On receipt of the mail, the certifier runs "PCA -dp".
```

```
Answer yes.

3.5 - Running "PCA -dr"
---------------
  The person who ran the RA, upon receiving the PEM
mail answer,  runs "PCA -dr". The certification data
are cached.The certification of the RA by the CA is
done.

Running PCA may be troublesome in the present version
due to the particular semantics of the CHIMAERA_CA_BAL
setting.
In these demo scripts we supposed that
 CHIMAERA_HOME=${HOME}/chimaera/release+
If this is not your case, you must hack a bit the profile
files
demo/w-pca/.profile.ca
demo/w-ra/.profile.ca
demo/w-ra/.profile.ra
in order to have PCA reach the proper mailbox.

PCA runs are useful only after certification mail reception.

4 - APPENDIX
--------
  Examples.

4.1 - CA Example
-----------
...
tic:.../jpg#0$CA -d
/u/tic/0/rodeo/jpg/chimaera/release+/etc/CA
current dir: /u/tic/0/rodeo/jpg/chimaera/release+/demo/w-ca
source  ./.profile.ca
Start On-Line Certification Authority
  (/u/tic/0/rodeo/jpg/chimaera/release+/etc/CA)?
```

```
with Dname  :  <C=FR,O=Inria,OU=ROCQUENCOURT>
for domain  :  <C=FR,O=Inria,OU=ROCQUENCOURT>
running as Policy Certification Authority
CHIMAERA_CA_BAL=./mbox
CHIMAERA_DNAME=<C=FR,O=Inria,OU=ROCQUENCOURT>
CHIMAERA_HOME=/u/tic/0/rodeo/jpg/chimaera/release+
CHIMAERA_UNAME=<C=FR,O=Inria,OU=ROCQUENCOURT>
Confirm? (y/n):
y
CHIMAERA : Your password :
CA : generation keys startup ...
CA : Write keys
chi_genk - chi_write_keys : open the file
chi_genk - chi_write_keys : write keys
chi_genk - chi_write_keys : write RC
CA : generate certificate startup ...
chimaera_ca : Create a connect multicast or UDP socket

CA waits connection
...

4.2 - GS Example
-----------
...
tac:.../jpg#0$GS -d
+ GS -d
/u/tic/0/rodeo/jpg/chimaera/release+/demo/w-gs
Start Local Secret Key Agent (GS)?
with Dname :
for domain :
CHIMAERA_HOME=/u/tic/0/rodeo/jpg/chimaera/release+
Confirm? (y/n):
y
chimaera_gs : Initialization
chimaera_gs : No core dump
chimaera_gs : Create a connect multicast or UDP socket
...
```

```
4.3 - Output from "API -d"
-----------------
...
/u/tic/0/rodeo/jpg/chimaera/release+/demo/w-gs
Start Test of Application Interface (API)?
with Dname :
for domain :
Confirm? (y/n):
Certification path request, using udp.
request completed (1).
Path is:
< userCertificate= <
message= <
serialNumber= ’2EAD13F1’H;
signature= <
algorithm= "iso member-body
                                            840 113549 1 1 2";
parameters= < n= NULL»;
issuer= <OU="ROCQUENCOURT", O="Inria",
                                 C="FR">;
validity= <
notBefore= 941025141929Z;
notAfter= 951025141929Z>;
subject= <OU="ROCQUENCOURT", O="Inria", C="FR">;
subjectPublicKeyInfo= <
algorithm= <
algorithm= "joint-iso-ccitt 5 8 1 1";
parameters= < i= 513»;
subjectPublicKey= ’00110000010010000000001001
0000010000000101111010100010001011110101000101111011000110110
0101111011101111111101000001111111011001110001110011000110001
1011100001111101100000011110110101010011001101101000010100000
0100100111111000001110110010000000101011100100100111111101110
1000111110100111011110100001000010010100111001001011010010101
1011001110011110011011001110001111101101011010101101010000100
1010011100000000101100110100001110111100001110110101011110100
```

```
0000101010100001000000010010101110100011000000001101110001010
1100110100010110001110100101100110100100000010000000110000000
10000000000000001’B»;
algorithm= < algorithm= "iso member-body 840 113549 1 1 2";
parameters= < n= NULL»;
signature= ’0000000101010001000101110111010101111100001100111
1110010000000010000000101100111010101101001001101000110010111
1101000011010011110010101000111001001101101010011110000111111
0010100110000110110100110010111000110111110000100001010011001
0001110111010111010110011111110101110111111001111111110010100
1100111000111010101000100110100000110000111011110000110011111
1111101010101000100110101110011100011101001100011111000011011
1111001111100001000100011000000110000100111011000000101010001
1110001110011001010000100001111011101111111001’B>;
theCACertificates= <
»


****************
*              *
*    SUCCESS   *
*              *
****************
        1

Certification path request, using tcp.
request completed (1).
Path is:
<
userCertificate= <
message= < serialNumber= ’2EAD13F1’H;
signature= <
algorithm= "iso member-body 840 113549 1 1 2";
parameters= < n= NULL»;
issuer= <OU="ROCQUENCOURT", O="Inria", C="FR">;
validity= < notBefore= 941025141929Z;
notAfter= 951025141929Z>;
```

```
subject= <OU="ROCQUENCOURT", O="Inria", C="FR">;
subjectPublicKeyInfo= <
algorithm= < algorithm= "joint-iso-ccitt 5 8 1 1";
parameters= < i= 513»;
subjectPublicKey= ’0011000000100100000000001001000001000000101
1110101000100010111101010001011110110001101100101111011101111
1111010000011111110110011100011100110001100011011100001111101
1000000111101101010100110011011010000101000000100100111111000
0011101100100000001010111001001001111111011101000111110100111
0111101000010000100101001110010010110100101011011001110011110
0110110011100011111011010110101011010100001001010011100000000
1011001101000011101111000011101101010111101000000101010100001
0000000100101011101000110000000011011100010101100110100010110
0011101001011001101001000000100000001100000001000000000000000
1’B»;
algorithm= <
algorithm= "iso member-body 840 113549 1 1 2";
parameters= < n= NULL»;
signature= ’00000001010100010001011101110101011111000011001111
11100100000000010000000101100111010101101001001101000110010111
1101000011010011110010101000111001001101101010011110000111111
0010100110000110110100110010111000110111110000100001010011001
0001110111010111010110011111110101110111111001111111110010100
1100111000111010101000100110100000110000111011110000110011111
1111101010101000100110101110011100011101001100011111000011011
1111001111100001000100011000000110000100111011000000101010001
1110001110011001010000100001111011101111111001’B>;
theCACertificates= <
»


****************
*              *
*    SUCCESS   *
*              *
****************
        2
```

```
Signature request, using udp.
request completed (1).


****************
*              *
*    SUCCESS   *
*              *
****************
        3

Signature request, using tcp.
request completed (1).


****************
*              *
*    SUCCESS   *
*              *
****************
        4

One certificate request, using udp.
request 5 completed (1).


****************
*              *
*    SUCCESS   *
*              *
****************
        5

One certificate request, using tcp.
request 6 completed (1).
```

```
****************
*              *
*    SUCCESS   *
*              *
****************
        6

Checking local signature - verification #1
Verification #1, ret =  1


****************
*              *
*    SUCCESS   *
*              *
****************
        7

chimaera_client - chi_checkSign : invalid signature

Checking local signature - verification #2
Verification #2, ret = -1


****************
*              *
*    SUCCESS   *
*              *
****************
        8

Tracking up -> Name = <OU="Grenoble", O="Inria", C="FR">
Tracking up -> Name = <O="Inria", C="FR">
Tracking up -> Name = <C="FR">
Tracking up -> Name = <OU="Grenoble", O="Inria", C="FR">
Tracking up -> Name = <O="Inria", C="FR">
Tracking up -> Name = <C="FR">
Tracking up -> Name = <OU="Muenchen", O="Siemens", C="DE">
Tracking up -> Name = <O="Siemens", C="DE">
Tracking up -> Name = <C="DE">
Tracking up -> Name = <C="DE">
chi_checkCertificate: - inverted hierarchy (Invalid argument)
Subject = <C="DE">
Issuer  = <C="DE">
Checking encryption - verification #9
cli crypt : asn1 field -(pkcs-pad)-> padded
Unencrypted text:
-----------
OctetString length: 35 octets
OctetString:

  4 21 54 68 69 73 20 69 73 20 73 6f 6d 65 74 68      .!This is someth
 69 6e 67 20 6e 61 74 75 72 61 6c 20 74 6f 20 73      ing natural to s
 69 67 6e                                             ign

Encrypted text:
----------
OctetString length: 64 octets
OctetString:

  0  1 ff ff ff ff ff ff ff ff ff ff ff ff ff ff      ................
 ff ff ff ff ff ff ff ff ff ff ff ff  0  4 21 54      ..............!T
 68 69 73 20 69 73 20 73 6f 6d 65 74 68 69 6e 67      his is something
 20 6e 61 74 75 72 61 6c 20 74 6f 20 73 69 67 6e       natural to sign

cli crypt : pkcs_padded -(rsa)-> cryptogram
Unencrypted text:
-----------
OctetString length: 64 octets
OctetString:

  0  1 ff ff ff ff ff ff ff ff ff ff ff ff ff ff      ................
 ff ff ff ff ff ff ff ff ff ff ff ff  0  4 21 54      ..............!T
 68 69 73 20 69 73 20 73 6f 6d 65 74 68 69 6e 67      his is something
 20 6e 61 74 75 72 61 6c 20 74 6f 20 73 69 67 6e       natural to sign

Encrypted text:
----------
OctetString length: 64 octets
OctetString:

 39 99 ce 93  9 ba ea f0 ca 7e c9 da 2a 7d a3 5f      9...........*.._
 46 81 bb 57 f8 41 90 e8  e da 16 f4 30  a ab ba      F..W.A......0...
 bf 17 5e 92 f8 32 8e 9b 4b 37 cb 64 b9  1 2f f8      ..^..2..K7.d../.
 2b 27 9f fd 4e 8b e7 9d c9 ea 79 b2 a8 2f 6f 80      +'..N.....y../o.

cli crypt : asn1 field obj -()-> cryptogram
Unencrypted text:
-----------
OctetString length: 35 octets
OctetString:

  4 21 54 68 69 73 20 69 73 20 73 6f 6d 65 74 68      .!This is someth
 69 6e 67 20 6e 61 74 75 72 61 6c 20 74 6f 20 73      ing natural to s
 69 67 6e                                             ign

Encrypted text:
----------
OctetString length: 64 octets
OctetString:

 39 99 ce 93  9 ba ea f0 ca 7e c9 da 2a 7d a3 5f      9...........*.._
 46 81 bb 57 f8 41 90 e8  e da 16 f4 30  a ab ba      F..W.A......0...
 bf 17 5e 92 f8 32 8e 9b 4b 37 cb 64 b9  1 2f f8      ..^..2..K7.d../.
 2b 27 9f fd 4e 8b e7 9d c9 ea 79 b2 a8 2f 6f 80      +'..N.....y../o.

Checking encryption -verification #9, ret =  1


****************
```

```
*               *
*    SUCCESS    *
*               *
****************
        9

 tcli: this is our cryptogram x 2
Unencrypted text:
-----------
OctetString length: 64 octets
OctetString:

 39 99 ce 93  9 ba ea f0 ca 7e c9 da 2a 7d a3 5f      9...........*.._
 46 81 bb 57 f8 41 90 e8  e da 16 f4 30  a ab ba      F..W.A......0...
 bf 17 5e 92 f8 32 8e 9b 4b 37 cb 64 b9  1 2f f8      ..^..2..K7.d../.
 2b 27 9f fd 4e 8b e7 9d c9 ea 79 b2 a8 2f 6f 80      +'..N.....y../o.

Encrypted text:
----------
OctetString length: 64 octets
OctetString:

 39 99 ce 93  9 ba ea f0 ca 7e c9 da 2a 7d a3 5f      9...........*.._
 46 81 bb 57 f8 41 90 e8  e da 16 f4 30  a ab ba      F..W.A......0...
 bf 17 5e 92 f8 32 8e 9b 4b 37 cb 64 b9  1 2f f8      ..^..2..K7.d../.
 2b 27 9f fd 4e 8b e7 9d c9 ea 79 b2 a8 2f 6f 80      +'..N.....y../o.

Checking decryption - verification #10
Checking decryption -verification #10, ret =  1


****************
*               *
*    SUCCESS    *
*               *
****************
        10
```

```
Unencrypted text:
-----------

 54 68 69 73 20 69 73 20 73 6f 6d 65 74 68 69 6e      This is somethin
 67 20 6e 61 74 75 72 61 6c 20 74 6f 20 73 69 67      g natural to sig
 6e                                                   n
Identity of objects: 0 (should be zero.)
Checking storage with validation#11


****************
*              *
*    SUCCESS   *
*              *
****************
        11

Checking certificate verification procedure #12


****************
*              *
*    SUCCESS   *
*              *
****************
        12

Checking storage with validation #13


****************
*              *
*    SUCCESS   *
*              *
****************
        13
```

```
Checking certificate verification procedure #14


****************
*              *
*    SUCCESS   *
*              *
****************
        14

Checking Secure certificate procedure #15


****************
*              *
*    SUCCESS   *
*              *
****************
        15

Checking Secure certificate procedure #16


****************
*              *
*    SUCCESS   *
*              *
****************
        16

(API) Test is terminated.

...

4.4 - Output of "GS -d" process upon API test
------------------------------
```

```
...

...chi_serv : process CA certification path
chi_serv : process CA certification path
chi_serv : process please sign
chi_serv : process please sign
chi_serv : process one certificate
chi_serv : process one certificate
chi_serv : process uncrypt this cryptogram
serv uncrypt: uncrypt <-(rsa)- cryptogram
Unencrypted text:
-----------
OctetString length: 63 octets
OctetString:

  1 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff      ................
 ff ff ff ff ff ff ff ff ff ff ff  0  4 21 54 68      .............!Th
 69 73 20 69 73 20 73 6f 6d 65 74 68 69 6e 67 20      is is something
 6e 61 74 75 72 61 6c 20 74 6f 20 73 69 67 6e         natural to sign

Encrypted text:
----------
OctetString length: 64 octets
OctetString:

 39 99 ce 93  9 ba ea f0 ca 7e c9 da 2a 7d a3 5f      9..........*.._
 46 81 bb 57 f8 41 90 e8  e da 16 f4 30  a ab ba      F..W.A......0...
 bf 17 5e 92 f8 32 8e 9b 4b 37 cb 64 b9  1 2f f8      ..^..2..K7.d../.
 2b 27 9f fd 4e 8b e7 9d c9 ea 79 b2 a8 2f 6f 80      +'..N.....y../o.

chimaera - chi_uncrypt_field : dubbious PKCS block...

serv uncrypt: unpadded <-(un-pkcs)- uncrypt
Unencrypted text:
-----------
OctetString length: 35 octets
OctetString:
   4 21 54 68 69 73 20 69 73 20 73 6f 6d 65 74 68      .!This is someth
  69 6e 67 20 6e 61 74 75 72 61 6c 20 74 6f 20 73      ing natural to s
  69 67 6e                                             ign

Encrypted text:
----------
OctetString length: 63 octets
OctetString:

   1 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff      ................
  ff ff ff ff ff ff ff ff ff ff ff  0  4 21 54 68      .............!Th
  69 73 20 69 73 20 73 6f 6d 65 74 68 69 6e 67 20      is is something 
  6e 61 74 75 72 61 6c 20 74 6f 20 73 69 67 6e         natural to sign

chi_serv : process please sign
chi_serv : process put certificate
chi_serv : process one certificate
chi_serv : process one certificate
chi_serv : process please sign
chi_serv : process put certificate
chi_serv : process one certificate
chi_serv : process one certificate
...
chi_serv : process one certificate
...
```

8.3 mailer

- *emacs/chimaera.el* defines the functions chimaera-crypt and chimaera-decrypt. Emacs-Lisp source code.
- *emacs/chimaera.emacs* text to be inserted in user's .emacs
- *mh/mh_profile* sample profile for MH. Instructs mh first to replace standard send and show by mysendproc and myshowproc then to set mhn's input filter
- *metamail/mailcap* xmh's settings to view PEM mails.

9 Mail, Integration within existing mailers

9.1 Using Chimaera from GNU Emacs

Here are the compatible tested versions for the commands chimaera-crypt and chimaera-decrypt:

- GNU Emacs 19.19.1
- GNU Emacs 18.58.0
- GNU Emacs 18.58.0
- GNU Emacs 19.6.1 Lucid
- Epoch 3.2.4
- Epoch 4.0.1

To operate these two functions, one must insert into its .emacs file the definitions of mailer/emacs/chimaera.emacs and must get a copy of the file chimaera.el Then get in mail mode ((Esc-X mail or Esc-X rmail). Once the mail is composed with a "Privacy:" header, one can apply the function chimaera-crypt with Esc-X ... or with the bind key C-cc. On input, the mail's origin can be checked with chimaera-decryp (C-cu).

Let us give a full illustration of GNU emacs/chimaera operation: Under GNU emacs, type C-x m and begin to compose the following message:

```
To: jpg
Subject:
Privacy:encode
-text follows this line-
secret
```

Then sign it by typing C-cc, the "chimaera crypt". You are prompted y/n for signing, depending of the security mode of the running chimaera_gs. You sign. Now you are asked it you want to send the encrypted mail: you say y...

On receipt, you go to read your mail thru M-x rmail and get into your emacs buffer the encrypted message:

```
Date: Thu, 1 Dec 1994 15:57:26 +0100
From: "J.-P. Giacometti" <Jean-Patrick.Giacometti\@sophia.inria.fr>
To: jpg\@mitsou.inria.fr


---BEGIN PRIVACY-ENHANCED MESSAGE---
Proc-Type: 4,MIC-ONLY
Content-Domain: RFC822
Originator-Certificate:
 MIIBUzCB/gIELpWLATANBgkqhkiG9w0BAQIFADAuMQswCQYDVQQGEwJGUjEOMAwG
 A1UEChMFSW5yaWExDzANBgNVBAsTBlNPUEhJQTAeFw05NDEwMDcxNzUzMDVaFw05
 NTEwMDcxNzUzMDVaMDwxCzAJBgNVBAYTAkZSMQ4wDAYDVQQKEwVJbnJpYTEPMA0G
 A1UECxMGU09QSElBMQwwCgYDVQQDEwNqcGcwWTAKBgRVCAEBAgICAANLADBIAkEA
 0Z/B7ugJGyjB66zEd5V++nXtdBI5UQHF8ibUgGuQZGtsWXFKs9lpwW28aTpN774t
 Co1HgyFCCw0Xtk1RLhdmWQIDAQABMA0GCSqGSIb3DQEBAgUAA0EAJ/GDLbnVbD5Z
 ORIlC7YHb1PdXXNH262CAiXLM4O3euW34cJxa8Qf9Dstq7njuD4ROlZ7PRKmOuLP
 vd508NBzyQ==
MIC-Info: RSA-MD5,RSA,
 txXBjb1xx91bGTVOO5SMky/hof9QsUlVnPmcSiEdxi0qoe/LzYNKhWRwUSLuGGn4
 /N8u9yw43NpaE7xWQG7sYg==

c2VjcmV0DQo=
---END PRIVACY-ENHANCED MESSAGE---
```

By typing C-cu, the "chimaera uncrypt", you get:

```
Date: Thu, 1 Dec 1994 15:57:26 +0100
From: "J.-P. Giacometti" <Jean-Patrick.Giacometti\@sophia.inria.fr>
To: jpg\@mitsou.inria.fr

Originator: <CN=jpg, OU=SOPHIA, O=Inria, C=FR>
-> Valid until Sat Oct  7 18:53:05 1995
Signature: OK

secret
```

9.2 Using Chimaera from MH

MH is the RAND Message Handling System. The use of the MH mailer commands "mhl" and "send" can be replaced for Chimaera purpose by the use of mysendproc and myshowproc.

MH version under consideration is 6.8.1.

The file .mh_profile should be augmented with the two lines:

```
showproc: myshowproc
sendproc: mysendproc
```

Parameters remain unchanged.

One can view an unencode PEM mail thru the viewing mh capability. Insert in .mh_profile the two lines:

```
mhn-show-application/pem-1421: pem\_show %f
mhn-show-application/pem: pem\_show %f
```

Releases file mh/mh_profile is a sample profile for MH. It instructs mh first to replace standard send and show by mysendproc and myshowproc then to set mhn's input filter.

9.3 Using Chimaera from XMH

XMH is the Weissman's X11 interface. Insert into your .mailcap file the two lines:

```
application/pem-1421; myshowproc %s
application/pem; myshowproc %s
```

This will ask for check when attempting to read a PEM message.

See released file metamail/mailcap for xmh's settings to view PEM mails.

10 Setting up Chimaera

10.1 Installation

Quick guide to the CHIMAERA Installation.

10.1.1 TO COMPILE

In directory src, execute:

```
make all
```

Warning: this will remove the binaries. Incremental compilation is done by:

```
make
```

10.1.2 TO COMPILE FOR K-GAM

```
make EXTRA=-DK\_GAM  all
```

Note that *make clean* should be issued only by sites having mavros compiler disposal. That will remove mavros generated files.

Note also that default compile options imply DEBUG.

10.2 Configuration

Argument "-*help*" displays a summary of the command. If you forget all the other options, remember this one.

Options are taken from the command line, from an ordered set of configuration files and from the environment.

Defaults values are systematically provided to minimize the amount of preliminary tuning. In that spirit, previous release options have been removed. The chimaera process is split into two distinct programs, *chimaera_gs* and *chimaera_ca*. The port used by *chimaera_gs* is now internally randomly set.

The environment variables are parsed before the configuration files, which are in turn parsed before the command line, so command line options override the configuration files options which in turn override the environment variables.

The sample file *etc/conf.sample.sh* can be mimicked – read and execute by a sh-like shell – to set thru the command environment the basic variables needed to run Chimaera.

Typically this information is created by the physical authority who is in charge of CA running. This information is then put at hand for potential users to enable proper GS running.

Lets's have a look at this sample file.

```
CHIMAERA_CAHOST="pax.inria.fr"
CHIMAERA_PORT="2001"
CHIMAERA_ALGO="md2WithRsaEncryption"
CHIMAERA_DNAME="<C=FR, O=Inria, OU=Sophia>"
CHIMAERA_UNAME="<C=FR, O=Inria, OU=Sophia>"
export CHIMAERA_CAHOST CHIMAERA_PORT CHIMAERA_ALGO
export CHIMAERA_DNAME CHIMAERA_UNAME
```

The CHIMAERA_CAHOST variable tells which host is supposed to run the CA; this is required in case the user host kernel does not know about IP multicast. Thus *chimaera_gs* process knows which CA host to contact.

The CHIMAERA_PORT variable tells *chimaera_gs* on which port the CA is listening.

The CHIMAERA_ALGO parameter specifies which algorithm – among

RSA

RSA-MD2

RSA-MD5

md2WithRsaEncryption

md4WithRsaEncryption

md5WithRsaEncryption

md2WithRsa

md4WithRsa

md5WithRsa

– is used to generate the keys and compute the Message Integrity Checks (MIC).

We recommend the use of md5WithRsaEncryption, as this algorithm has been thoroughly tested.

Note that the above parameters have received defaults. This is not the case for the following two, whose explicit setting is compulsory.

The CHIMAERA_DNAME holds the "Distinguished Name". This identifies the CA without ambiguity.

The CHIMAERA_UNAME yields the CA's "Base Name" in X500 parlance.

In CHIMAERA_DNAME="<C=FR, O=Inria, OU=Sophia>" , C tells the country – France –, O the organization – Inria – and OU finally tells the particular unit – Sophia Antipolis lab – which forms the domain of concern.

The syntax of CHIMAERA_UNAME is alike.

It is convenient to have the basic Chimaera settings in your session shell initialisation file.

10.3 CA

The command *chimaera_ca* starts the CA process. You must then provide your password. Note that the CA must not be run as a background task. The CA then builds its own certificate – that may take a while.

In case IP multicast support is lacking, this warning follows:

```
"chimaera - ca : ip Multicast disable (Invalid argument)"
```

Upon message

```
"CA waits connection"
```

the CA is ready to operate.

So far, two files have been created in the Chimaera directory, whose default is the current working directory.

The file *.chimaera.key* holds the private key of the CA (in fact the guy who is running it). This key is encrypted with the password.

The file *.chimaera.db* is the repository holding the certificates. At launch, it contains merely the CA's certificate. It then receives the copy of the certificates clients (other *chimaera_gs* processes) asked for.

A CA process is intended to run permanently as a community server and certificate provider.

10.4 GS

The command *chimaera_gs* starts the GS process. When run first time, the GS process contacts the CA either thru an IP multicast call or an IP call to the host specified.

At subsequent runs, the GS uses the files *.chimaera.key* and *.chimaera.db* it created at first time.

Once a user has got its certificate, he does not need the CA any more. In fact its local cache has two entries:

- its own certificate holding its public key.
- the certificate of the CA that certified the former.

The GS is operative as soon as the message

```
"GS : Save certificate"
```

appears. There comes the Chimaera icon. Clicking into the Chimaera icon with Left Button shows remaining time before next password explicit check. Middle and Right Button pop a menu enabling to change current validation mode among "Automatic," "Confirm" and "Check". Note that default mode is Confirm. The validation mode setting allows signing to occur silent (Automatic), or to require a confirmation click (Confirm) or to require re-entering the user password (Check). When in Check mode, confidential data are not held in memory more than needed.

As an additional security wall, during a Chimaera session, a key is build and made compulsory to enable an application's service. This key is held in the file .chimaera.session. Default is to establish this file into the current directory for session time.

Another security feature for Chimaera under X Window System is to reject any application call from outside the local host.

10.5 Exchanging Privacy Enhanced MAIL

Put the *pem_create* and *pem_show* commands into your user PATH. These work as Unix-filters (taking input on stdin stream and forwarding output to stdout stream) or with arguments (-*i* and -*o*).

```
Examples:

pem_create < PEM_mail > Signed_PEM_mail

pem_create < mail_pem | pem_show

pem_create -i PEM_mail -o Signed_PEM_mail

pem_show -i Signed_PEM_mail -o Checked_PEM_mail
```

11 Implementation

11.1 User Names (Unix and the X500 Directory)

This is to begin to elucidate the question dated #1 Date: Fri, 17 Feb 95 10:55:50 WET

« After user key generation we examined the db on both the CA and the GS and we saw that the "subject" name includes a 'CN="olivet"'field that we did not set in the environment (we only have Is it a mistake of ours not to set the "CN=" ? I don't think we have "olivet" written anywhere in our system. Where is the chimaera taking this string from ? Is it truncated to 6-chars for some reason ? »

The name of the user (COMMON_NAME in X500 slang) is known throughout chimaera as:

```
chi_param.c:71:char                     username[MAXNAMELEN];
```

If a user name is found in the global environment(profile < env < args), it is copied at:

```
chi_param.c:514:        strcpy(username, parm_value[7]);
```

Then a check is made to see if it is a known user name with the *getpwnam* routine. If the global environement is mute about the name of the user, the *uid* of this process is tried as a reasonable hint with the *getpwuid* routine.

Is there in 'CN="olivet"' field the result of the *getpwuid*? That is a point to fix. Are you running the gs as user 'olivet' ?

As for the size, the maximum is considered to be 8 bytes.

11.2 OID (The object identifier concept)

The implementation is from asn1.h:

```
typedef asn1_field asn1_oid;
```

11.3 Algorithm Name (The name used by humans)

The algorithm name specifies both the encoding method (e.g. RSA) and the hashing function (e.g. MD5), e.g. "md2WithRsaEncryption".

11.4 Algorithm Object

```
typedef struct alg_desc_type {
  asn1_oid      algo_oid;
  int           algo_param;     /* -1: NULL, >= 0: Parameter Value */
  int           (* hash_func) ();
  int           value;
  char          *name;
} alg_desc_type;
```

oid algorithm definitions

```
/* from chimaera.h */
#define SECSIG_OID                 43,14,3
#define SECSIG_ALGO_OID            SECSIG_OID,2
#define SECSIG_SIGN_ALGO_OID       SECSIG_ALGO_OID,3

#define DSSIG_OID                  43,14,7
#define DSSIG_ALGO_OID             DSSIG_OID,2
#define DSSIG_ENCRYPT_ALGO_OID     DSSIG_ALGO_OID,1
#define DSSIG_HASH_ALGO_OID        DSSIG_ALGO_OID,2
#define DSSIG_SIGN_ALGO_OID        DSSIG_ALGO_OID,3

v#define DS_OID                     85
#define DS_ALGO_OID                DS_OID,8
#define DS_HASH_ALGO_OID           DS_ALGO_OID,2
#define DS_ENCRYPT_ALGO_OID        DS_ALGO_OID,1
#define DS_SIGN_ALGO_OID           DS_ALGO_OID,3

#define PKCS                       42,134,72,134,247,13
#define PKCS_1                     PKCS,1,1

/* algo internals */

unsigned char md4WithRsa_oid_val    [] = {SECSIG_ALGO_OID,2};
unsigned char md5WithRsa_oid_val    [] = {SECSIG_ALGO_OID,3};
unsigned char md4WithRsaEncr_oid_val[] = {SECSIG_ALGO_OID,4};
unsigned char md2_oid_val           [] = {DSSIG_HASH_ALGO_OID,1};
unsigned char md2WithRsa_oid_val    [] = {DSSIG_SIGN_ALGO_OID,1};
unsigned char elGamal_oid_val       [] = {DSSIG_ENCRYPT_ALGO_OID,1};
unsigned char md2WithElGamal_oid_val[] = {DSSIG_SIGN_ALGO_OID,2};
unsigned char rsa_oid_val           [] = {DS_ENCRYPT_ALGO_OID,1};
unsigned char sqModn_oid_val        [] = {DS_HASH_ALGO_OID,1};
unsigned char sqModnWithRsa_oid_val [] = {DS_SIGN_ALGO_OID,1};
unsigned char rSAmd2WithRSA_oid_val [] = {PKCS_1,2};
unsigned char rSAmd4WithRSA_oid_val [] = {PKCS_1,3};
unsigned char rSAmd5WithRSA_oid_val [] = {PKCS_1,4};
unsigned char rsa_md5               [] = {PKCS,2,5};
unsigned char rsa_md2               [] = {PKCS,2,2};
```

11.5 Algorithm Objects Table

```
alg_desc_type alg_desc_table [] = {
 { {4, rsa_oid_val,           0}, 512, (int (*)())0, RSA,      "RSA"},
 { {8, rsa_md2,               0},  0,  md2_hash,     RSA_PKCS, "RSA-MD2"},
 { {8, rsa_md5,               0},  0,  md5_hash,     RSA_PKCS, "RSA-MD5"},
 { {9, rSAmd2WithRSA_oid_val, 0},  0,  md2_hash,     RSA_PKCS, "md2WithRsaEncry
 { {9, rSAmd4WithRSA_oid_val, 0},  0,  md4_hash,     RSA_PKCS, "md4WithRsaEncry
 { {9, rSAmd5WithRSA_oid_val, 0},  0,  md5_hash,     RSA_PKCS, "md5WithRsaEncry
 { {6, md2WithRsa_oid_val,    0}, -1,  md2_hash,     RSA,      "md2WithRsa"},
 { {5, md4WithRsa_oid_val,    0}, -1,  md4_hash,     RSA,      "md4WithRsa"},
 { {5, md5WithRsa_oid_val,    0}, -1,  md5_hash,     RSA,      "md5WithRsa"},
};

int nb_algo_oid = sizeof(alg_desc_table)/sizeof(alg_desc_type);
```

Some current index values

```
#define CHI_ALGO_RSA 0
#define CHI_ALGO_RSA_MD2 1
#define CHI_ALGO_RSA_MD5 2
```

11.6 Algorithm Interface

11.6.1 How to handle an algorithm

The function get_algo_by_name gets the internal descriptor of the algorithm from its external name.

```
int get\_algo_by_name(algo_name, ix)
     char *algo_name;
     int  *ix;
{
  for (*ix=0; *ix < nb_algo_oid; (*ix)++)
    if (ASN1_CMP(algo_name, alg_desc_table[*ix].name,
                 strlen(algo_name)) == 0)
      return(*ix);
  return(-1);
}
```

11.6.2 How to exchange an algorithm

How to get the exchange form of an algorithm.

```
    AlgorithmIdentifier_err(&ca_algo);/* from chi_param.c */
    if (get_algo_by_name(parm_value[3], &indx) < 0) {

    }else{
        asn1_field_cpy(&(ca_algo.algorithm),
                         &(alg_desc_table[indx].algo_oid));
        switch (alg_desc_table[indx].algo_param) {
        case -1 :
            ca_algo.parameters.x = -1;
            ca_algo.parameters.v.n = 0;
            break;
        case 0:
            ca_algo.parameters.x = 2;
            ca_algo.parameters.v.n = 0;
            break;
        default:
            ca_algo.parameters.x = 1;
            ca_algo.parameters.v.i =
                           alg_desc_table[indx].algo_param;
            break;
        }
    }
```

11.6.3 How to use an algorithm

What does happen in module chi_tcli? (initial wrong typed version)

```
...
   /*
     * Then, we test the effectivity of local signatures.
     */
    for (i=0; i<2; i++, invoke_id++){
        printf("Signature request, using %s.\n", protocol[i]);
        ret = chi_pleaseSign
            (protocol[0], appliname, dirname, username,
             invoke_id, (void *)&tobesigned, Ostring_cod,
     Ostring_len,
             &alg_desc_table[CHI_ALGO_RSA_MD5], &signature);
        printf("\trequest completed (%d).\n", ret);
        if (ret == 1){
            tag_print(invoke_id, True);
        }else{
             tag_print(invoke_id, False);
            exit(2);
        }
        fflush(stdout);
    }
...
```

What does function chi_pleaseSign do with this algorithm information? It puts it into the request which will be dispatch via chi_rpc to the function process_pleaseSign. The latter uses it into a chi_sign_field call.

The "chi_sign_field" procedure signs a field using the local private key. This is done by calling first a hash procedure, then a selected encryption algorithm. mind: this func can only be called by server, which needs local key. key_info is a useless argument.

```
   /* Get algorithm identifier */
    if ( (algo_id == (AlgorithmIdentifier *)0) ||
        (get_algo_by_oid(&(algo_id->algorithm),
                         &indx) == -1) )
   {
           err_ret("chimaera - chi_sign_field : invalid\
     algorithm identifier");
           return(-1);
    }

    /* Hashing */
    if (alg_desc_table[indx].hash_func(asnin, &asnout) < 0)
   {
           err_ret("chimaera - chi_sign_field : hash \
    function failed");
           return(-1);
    }

...
```

From this one sees that chi_tcli holds a correct value but typing is wrong.

```
type AlgorithmIdentifier            type alg_desc_type
        asn1_filed algorithm;            asn1_oid algo_oid;
```

This works because

1. asn1_field === asn1_oid
2. The first action is to set the correct algo_oid into algorithm.

Sigh!

11.7 Algorithm Identifier

Algorithm specifications are not only vehiculated by humans, in external form, and by programs, the internal implementation, but also by networks, the exchange form.

```
typedef struct AlgorithmIdentifier {/* from CHIMAERA.h */
        asn1_field algorithm;
        struct {
                int x;
                union {
                        asn1_int32 i;
                        int n;
                } v;
        } parameters;
} AlgorithmIdentifier;
```

References

[1] Anas Altarah, *Chimaera: un modèle pour la securité des systèmes ouverts* , Thèse de l'Université de Nice, 1992.

[2] Pierre Pacchionni, *Rapport de stage*, personnal communication.

[3] Étienne Debroucker, Anthony Charles, *Extension des services de securité dans le serveur Chimaera*, Rapport ESSI, Université de Nice, 1993.

[4] Aline Russeil, *Chimaera – Étude et améliorations*, Rapport DESS-ISI, Université de Nice, 1994.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399