Implementation of Strong Authentication in CASTOR

Benjamin Couturier,
European Organization for Nuclear Research

Vitaly Motyakov,
Institute for High Energy Physics, Russia

Revisions

Revision	Date	Comment
0.1	08/23/04	First Draft
0.2	09/01/04	Update with plugin documentation
0.9	09/02/04	Included reviews from VM
1	09/16/04	Included comments from CASTOR Team

Table of content

CASTOR, **C**ERN's **A**dvanced **STOR**age Manager is the hierarchical storage manager currently in use at CERN. It is based on SHIFT, a very successful tape management system built in the 1990s. Both projects focused on providing the physics community with the functionalities needed to record and exploit physics data from the experiments. Having a strong security model was not amongst the primary requirements for physics applications; however, due to the success of SHIFT and CASTOR, more and more user files are now also stored in the system, raising the issue of proper authentication and security in CASTOR. Furthermore, the Grid initiatives will allow the sharing of data all over the globe, adding new security requirements.

Based on the proposal for improving the CASTOR security (c.f. CASTOR_Security.doc) a security plugin was implemented and integrated with the various CASTOR components. The goal of this document is to present the work that was done, what issues were encountered and what the next steps are towards a secure CASTOR. Please note that the security API is still a work in progress, and the main issues mentioned in this document will have to be solved before a secure CASTOR implementation can be deployed.

1 CASTOR Security Plugin

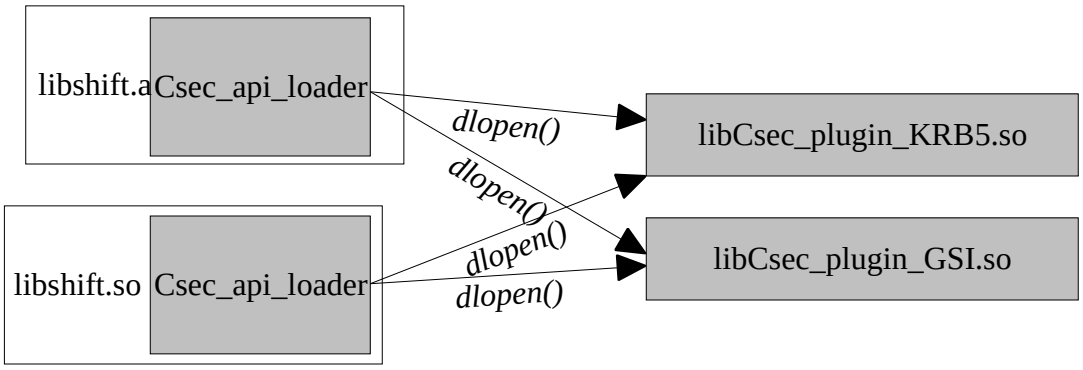
1.1 Goals and requirements

The goal of the security plugin is to ensure that strong security can be implemented easily in the various CASTOR components. The API must be simple and hide the complexity of the security mechanims.

The security system must be flexible enough for allow for adding new mechanism and allow a server to access requests authenticated with different mechanisms at the same time.

1.2 Architecture

1.2.1 Plugin loader



A new module has been added to the CASTOR distribution called “security”. In this directory can be found the code allowing to implement authentication in CASTOR.
The security module consists of several parts:

- A common security API, compiled into libshift.a and libshift.so which contains the code allowing to load the various security modules
- A set of security modules, implemented as stand-alone shared libraries.

The code in place allows to load the plugin necessary for the mechanism decided by the negociation that takes place between the client and the server, before the actual authentication is performed.

The current way of loading the plugin is to use the dynamic loading library, notably the calls *dlopen()* and *dlclose()*.

N.B. 1 The use of dynamic loading of shared functionality in the security plugin, linked into *libshift.a* forces the programs using libshift.a to specify the "-ldl" linker option !

N.B.2 The fact that the security plugin used *dlopen* to access the plugin functionality can create some problems for programs that link statically with libshift.a and use CASTOR utilities such as Cglobals: static variables in those modules have different values in the statically linked binary (as it linked from libshift.a) than in the security plugin which uses libshift.so ! This is a source of problems for the Security layer, as it stores the error messages in a per thread variable: that variable, filled in by the plugin is not visible by the program linked with libshift.a ! There are several solutions to solve this problem:

- Always link libshift dynamically and abandon static linking in CASTOR.
- Try and compile the security modules statically. Some work has been done in that direction, but it is not possible to build the GSI and KRB5 plugin together as they depend on two implementations of the GSSAPI library, with the same method names...
- Separate the security plugin, and create a separate shared library with no static equivalent. It would be linked with libshift.so, like the plugins and they would share the same Cglobals object.
- Store the CASTOR security layer errors in a the security context rather than in a per-thread variable. This is cleaner and should be done anyway !

1.2.2 Security plugins

The security modules for the following mechanisms have been implemented for the moment:

- GSI: The Globus Security Infrastructure (using the GSSAPI)
- Kerberos 5 (using the GSSAPI)
- Kerberos 4 (for compatibility while Kerberos 5 is not un full production at CERN)
- ID (an unsafe protocol passing the uid/gid, to be used for tests only)

1.2.3 CASTOR Security API

All declarations necessary in order to use can be found in the "Csec_api.h" header file. Csec_api functions all use a security context, of type *Csec_context_t* which must first be created and initialized.

Client API

On the client, the *Csec_client_init_context* must be called to initialize the security context. It takes as argument:
- A pointer to the security context to be initialized
- The service type of the server which is going to be contacted using this context (see 1.2.4)
- A NULL terminated array of *Csec_protocol* objects.

There are several ways of configuring the security layer so as to specify which mechanism should be used when communicating with the server:
- If a *Csec_protocol* list is specified in *Csec_client_init_context*, these values will be taken into account
- If the *Csec_protocol* list is NULL, buf if the CSEC_MECH environment variable is defined, then the value of CSEC_MECH will be used
- otherwise, the default specified in site.def will be used.

Once the client context is initialized properly, and once the TCP connection between client and server is established, the authentication can be performed using the *Csec_client_establish_context()* API function. It takes as parameter a pointer to the security context, and the file descriptor corresponding to the TCP connection.
If it returns an error, the error number is stored in the “serrno”, a standard CASTOR feature, and the full error message can be retrieved with the *Csec_geterrmsg()* function.

Server API

On the server, the *Csec_server_init_context* must be called to initialize the security context. It takes as argument:
- A pointer to the security context to be initialized
- The service type of the server (see 1.2.4)
- A NULL terminated array of *Csec_protocol* objects.

There are several ways of configuring the security layer so as to specify which mechanism should be accepted when receiving a connection from a client:
- If a *Csec_protocol* list is specified in *Csec_server_init_context*, these values will be taken into account, and the server will only use the mechanisms specified in this list.
- If the *Csec_protocol* list is NULL, buf if the CSEC_AUTH_MECH environment variable is defined, then the value of CSEC_AUTH_MECH will be used
- otherwise, the default specified in site.def will be used.

Once the context is initialized properly, and once the TCP connection between the client and server is established, the authentication of the client can be performed the *Csec_server_establish_context()* API function must be called. It takes as parameter a pointer to the security context, and the file descriptor corresponding to the TCP connection established between client and server.
If it returns and error, the error number is stored in the "serrno", a standard CASTOR feature, and the full error message can be retrieved with the *Csec_geterrmsg()* function.

Once the security context is established, the identity of the client can be be known by using *Csec_server_get_client_username()*. This functions maps the client DN in the case of GSI, or the principal in the case of KRB5 to a local user on the machine.

N.B. Specifying the accepted mechanism in the environment variable is more flexible than specifying it in the *Csec_xxx_init_context()* call, that approach should be favored unless the application has other ways to configure the mechanisms, or if one and only one security mechanism should be used.

Common functionality

Once the context has been established and not needed any more, it is necessary to call *Csec_clear_context()* to clear the memory allocated in the context.

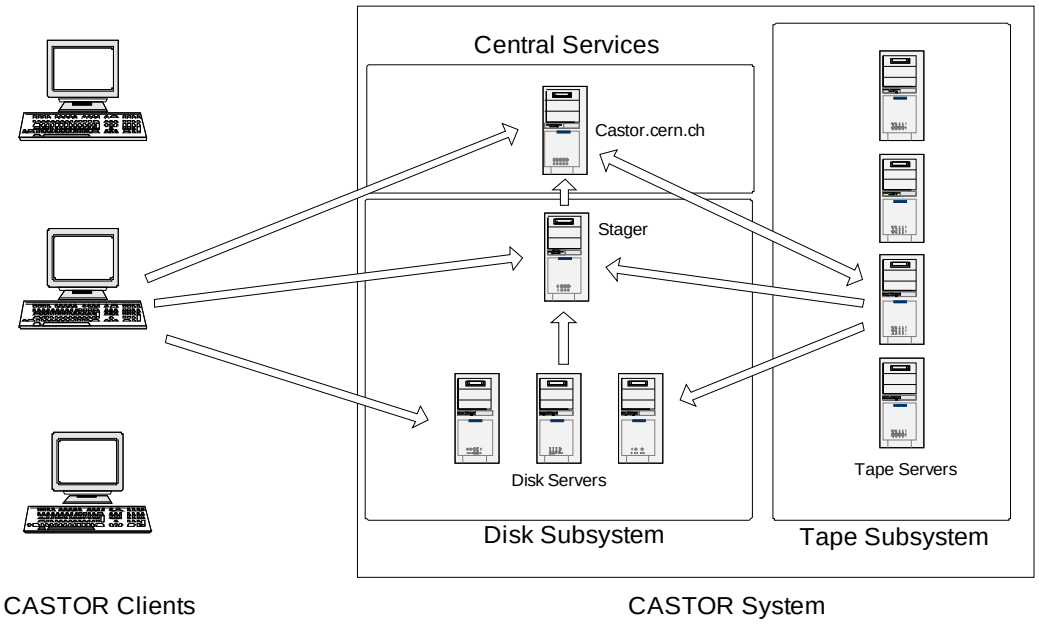
A mechanism has been put in place to allow tracing the security layer independently of the CASTOR services using it:
Setting the environment variable CSEC_TRACE to 1 causes the security layer to dump the packets exchanged during the security handshake, as well as debugging information about the behaviour of the plugin to stderr.
If CSEC_TRACEFILE is defined and contains a valid file name, the trace is sent to that file (this is practical when tracing a daemon).

1.2.4 Service Types/Service names

So as to manage the different type of service keys necessary for the CASTOR servers, the security layer uses the notion of service type. Three types of services have currently been defined in CASTOR, in the Csec_api header file (plus the host service type, corresponding to existing host keys which is accepted during the test phase).

```
#define CSEC_SERVICE_TYPE_HOST    0
#define CSEC_SERVICE_TYPE_CENTRAL 1
#define CSEC_SERVICE_TYPE_DISK    2
#define CSEC_SERVICE_TYPE_TAPE    3
```

Service Type	Meaning
CSEC_SERVICE_TYPE_CENTRAL	To represent all central servers, like name server, volume manager...
CSEC_SERVICE_TYPE_TAPE	For the tape daemon and the rtcopy server.
CSEC_SERVICE_TYPE_DISK	for the stager and rfio daemons.
CSEC_SERVICE_TYPE_HOST	Represents the host key. Should not be used by CASTOR in the futurebut is accepted for the moment.



The keys corresponding to the various service types, for the GSI and Kerberos 5 mechanisms, for the CERN installation, are detailed in the following table:

Service Type	*KRB5*	*GSI*
CSEC_SERVICE_TYPE_CENTRAL	**castor-central/<hostname>@CERN.CH**	**[...] CN=castor-central/<hostname>**
CSEC_SERVICE_TYPE_TAPE	**castor-tape/<hostname>@CERN.CH**	**[...] CN=castor-tape/<hostname>**
CSEC_SERVICE_TYPE_DISK	**castor-disk/<hostname>@CERN.CH**	**[...] CN=castor-disk/<hostname>**

2 Secured Applications

2.1 Authentication

The following applications have currently been modified to use the CASTOR security plugin:

- **tpdaemon**
- **vmgrdaemon**
- **Cupvdaemon**
- **vdqmserv**
- **rfiod**
- **rtcpd**
- **nsdaemon**
- **stgdaemon**.

All modifications were done to the current production castor components; we tried to modify new prototype components (and actually this is the case for the new **rfio** and **rtcopy)** but we encountered problems with the new stager: at the moment as it is still in development and is not mature enough; we decided to secure it at a later stage.

Secured application differs from standard one in a sense that the connection is authenticated. The authentication can be done with Kerberos5 or GSI. It means that both sides, client and server, must have valid Kerberos5 tokens or GSI certificates.

2.2 Authorization

Once the security layer is integrated in the various CASTOR services, the daemons can know the identity of the clients (by using the proper Csec_api calls) but the issue of authorization must be solved. The client is identified by a DN in the case of GSI, a principal in the case of KRB5, which is mapped to a local user and finally a set of uid/gid.

In CASTOR, the identity of the client was traditionally sent as the couple (uid/gid) in the various protocols. So as to keep compability with the CASTOR protocols, we decided to keep the uid/gid passed in the CASTOR protocol and consider them as an “authorization

ID", with the same meaning as in in the Simple Authentication and Security Layer (SASL)

Here is the definition from RFC 2222:

```
"During the authentication protocol exchange, the mechanism performs
authentication, transmits an authorization identity (frequently known
as a userid) from the client to server, and negotiates the use of a
mechanism-specific security layer.  If the use of a security layer is
agreed upon, then the mechanism must also define or negotiate the
maximum cipher-text buffer size that each side is able to receive.

The transmitted authorization identity may be different than the
identity in the client's authentication credentials.  This permits
agents such as proxy servers to authenticate using their own
credentials, yet request the access privileges of the identity for
which they are proxying.  With any mechanism, transmitting an
authorization identity of the empty string directs the server to
derive an authorization identity from the client's authentication
credentials."
```

In the daemons we now have two sets of uid/gid for a client request:

- One returned by the security layer, which is the authenticated identity of the client
- The second passed by the client in the API, which specifies which identity it wants to assume.

The implementation of authorization in the CASTOR daemons differs according to the server, but the following rules have been enforced:

- If the client is another server, that is if its DN or principal corresponds to a service name (c.f 1.2.4) (this can be done using *Csec_get_service_type()*) then the authorization uid/gid can be trusted.
- If the client is a normal user, then the authorization uid/gid are anyway replaced with those retrieved by the security layer if they differ.

2.3 Issues

2.3.1 Applications using setuid/setgid

Some daemons in CASTOR, namely **rfiod**, **stgdaemon** and **rtcpd** run as root and use setuid/setgid to have the the same uid/gid as the use who has made the request. This is a problem, as, as soon as the call to setuid/setgid has been done, the process cannot access the ticket file or keyfile, and cannot identify itself as a CASTOR server.

One solution to this would be to use extensions to the GSSAPI, *gss_export_cred()* and *gss_import_cred()* which exist in GSI, but unfortunately not in the Kerberos 5 implementations! (however they should now be integrated in the standard). We tried to use the GSSAPI doing the *gss_acquire_cred()* as root, and then doing the *gss_accept_sec_context()* after the calls to setuid/setgid but that approach does not work, as the Kerberos 5 implementations try to access the ticket file during the establishment of the security context !

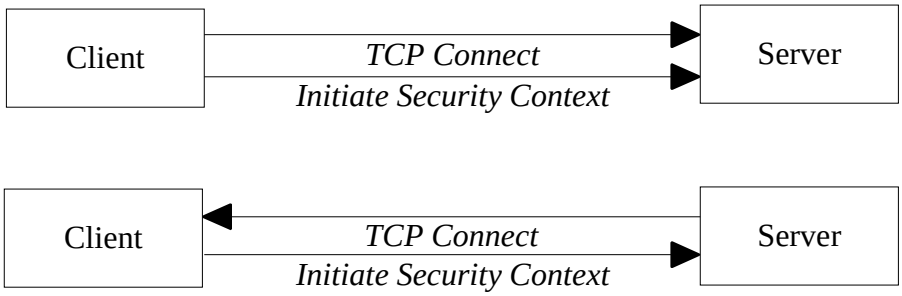
This is why we had to remove all calls to 'setuid' and 'setgid' from stgdaemon and rtcpd just to allow them to acquire server credentials. As a side effect we have now all entries in the name server database owned by root ! this is obviously not acceptable, and the various APIs willl have to be modified to allow the servers to specify the identity of the user on behalf of which they are working.

NB: This is a problem for the stager and RTCOPY but not for RFIO: in RFIO the security handshake is done before the setuid/setgid. However, at the moment only the control connection is secured, not the data connection and we would certainly hit the setuid/setgid problem while trying to authenticate the data connection !

2.3.2 Callbacks in RTCOPY

While integrating the security layer in the RTCOPY daemon, we faced the following problem: in the case of a “simple” connection the client opens the connection and initiates the security context (by calling *Csec_client_establish_context()*). The server accepts the TCP connection and calls *Csec_server_establish_context().*

In the case of RTCOPY, we have a callback mechanism: in this case the server connects to the listening port on the client side. The problem is that the client probably does not have the proper credentials for accepting a security context ! This is why the client has to accept the connection and immediately initiate the security context



N.B. This mechanism will be used heavily in the new CASTOR stager in which all calls made to the stager involve a callback from the daemon.

2.4 Credential delegation

In the current implementation of the CASTOR security layer, for inter-server communications, we decided to use service keys as client keys.

Another option, maybe more secure, would have been to use the delegation mechanisms, available in the GSSAPI libraries anyway.
However, we found that the delegation was not fulfilling our purpose anyway, as many of the inter-server CASTOR requests do not correspond to any client request: for example a migration is started by the stager when it decides to do so: the credentials delegated by the client might have expired at that point. Furthermore, it migrates many files belonging to many different users, so it would have to keep the delegated credentials of all those clients... This seems difficult to do in a first stage and we decided to abandon that approach in a first phase.

3 Configuration

The secured applications use TCP ports which differ from those used by the standard applications. The secured port can be specified in different ways:

a) by setting an environment variable
b) by specifying it in the castor configuration file
c) by specifying it in /etc/services.

If none of these methods is used then port will be set to the default value specified at the time of compilation. When specifying a secure port you must use the name similar to one used with the standard application but prefixing it with the letter 's' (“secure”). For example, to specify secure rfio port you must write:

a) when specifying the port through the environment variable:
export SRFIO_PORT=5501
b) when specifying the port in the configuration file:
SRFIO PORT 5501
c) when specifying the port in the /etc/services:
srfio 5501/tcp

The secured client applications can connect to insecure servers. By default they use insecure connections. To force them to use secure connections you must set the environment variable SECURE_CASTOR, for example:
export SECURE_CASTOR=1

The daemons (servers) don't have such flexibility, and can be set to be either secured or insecured at compile time.

3.1 Init scripts

The init.d scripts for the CASTOR servers can be found in the CASTOR distribution. In each module, the file <servername>.scripts should be placed in the **/etc/init.d** directory and renamed to <servername>. It is a script used to start/stop the corresponding daemon/server.
The file <servername>.sysconfig should be placed into the **/etc/sysconfig** directory and renamed to <servername>. This file is used during server startup.

There is also a file common to all secured servers. It is located in the **'security'** subdirectory and named **security.sysconfig**. This file should be moved to **/etc/sysconfig/castor_security**.

After all this is done, on Linux, you can use 'chkconfig' to add the new service to the system. For example, for the 'vmgrdaemon' yo will have the following files:

/etc/init.d/vmgrdaemon - the startup script;
/etc/sysconfig/vmgrdaemon - the configuration file;
/etc/sysconfig/castor_security – the common configuration file.

3.1.1 GSI – installing keys/certificates/map file

To use secured server with GSI you have to obtain the certificate for this particular service. How to generate test keys and certificates is described in Appendix A. Suppose you already have the host key and the host certificate. The default locations for them are as follows:

Host key:	/etc/grid-security/hostkey.pem
Host certificate:	/etc/grid-security/hostcert.pem
CA certificates:	/etc/grid-security/certificates

Note: The host key file MUST belong to the user under which the server program will run and MUST have only user read/write permissions, i.e. you have to change acces modes for this file to 600, otherwise the host key will be refused.

You can specify the host key and host certificate locations by using the following environment variables:

X509_USER_KEY	- host key file;
X509_USER_CERT	- host certificate file;
X509_CERT_DIR	- CA certificates.

You have to create GSI map file, which will contain mapping from distinguished names to local user names. It must contain one line per distinguished name in the form

<DN><sp><local_user_name>

where <sp> is any number of white space characters, for example:

"/OU=cern.ch/CN=Vitali MOTYAKOV" motiakov

The path to the file should be **/etc/grid-security/grid-mapfile**

3.1.2 KRB5 – Adding keys to keytab

This can be done with the **kadmin** comand. The principal must first be created and then exported to the keytab.

Once this is done, the server with access to the keytab can accept security contexts, but it still cannot initiate them ! To do so, it has to call **kinit -k <principal>**.
Of course, this ticket can expire and has to be renewed on a regular basis.

4 Building the secure CASTOR

The ***site.def*** file located in the CASTOR/config directory contains the configuration options for the CASTOR security layer. This file must be edited so as to build a version of CASTOR taylored for the needs of a site.

To enable the security layer in CASTOR, the BuildSecurity option should be switched on by specifying:

```
#define BuildSecurity          YES
```

A set of switches allows to specify for which security mechanisms the plugins should be built, the choice is:

- UseKRB5
- UseGSI
- UseKRB4

The Kerberos 4 module was implemented for compatibility and testing, however it is not recommended using it now as Kerberos 5 provides much better security.

The default chosen implementation of Kerberos 5 is the MIT Kerberos 5 (http://web.mit.edu/kerberos/www/) but it is also possible to build the Kerberos 5 plugin with the Heimdal Kerberos 5 library (http://www.pdc.kth.se/heimdal/), using the UseHeimdalKRB5 option.

There is also a set of switches, that allows to specify, application by application,which one should be built with the security layer: For example, if you want to build secured vdqm, you must define 'BuildSecureVdqm' as 'YES'. Note that you must define also 'BuildVdqmClient' and/or 'BuildVdqmServer' as 'YES'. You can also build static secure library. To do this you have to set 'SecMakeStaticLibrary' to 'YES'. In this case you will be able to use simultaneously only one authentication mechanism from the list of GSI, KRB5 or KRB4 due to clashes of library and function names. So, you have to choose which mechanism to use in 'site.def'.

The Kerberos 4 and 5 libraries are built using the krb[45]-config binaries provided with the distributions, which should allow not having to modify the site.def for future versions of the Kerberos distributions.

5 Migration to a secure CASTOR

Migrating an existing CASTOR installation, especially one the size of CERN, will be a difficult task. A detailed plan will be necessary to migrate without problems. For the moment we have taken the following design decisions, to try to ease the migration.

5.1 Server listening ports

The listening ports of the secure daemons, have been changed, so that, in the mixed mode (secure/insecure) period, it should be easy to know whether a server is running in secure mode or not.

The following table lists the secure ports for the daemons that have been secured.

Host	Daemon	Port
Central Machine	Name Server	TCP/5510
Central Machine	VDQM	TCP/5512
Central Machine	VMGR	TCP/5513
Central Machine	CUPV	TCP/5520
Disk Server	rfiod	TCP/5501
Tape server	rtcpd	TCP/5503
Tape server	tpdaemon	TCP/5511

5.2 Client compatibility

To allow for easier deployment, the secured castor clients can work in normal/insecure mode as well (and this is actually the default for the moment).
An environment variable (“CASTOR_SECURE”) switches them to secure mode, when it is defined.

6 Issues and further developments

The following problems have been found during the tests done on the secure version of CASTOR.

1. The Imakefile for the security module have to be modified. At the moment shared security libraries are not moved to **/usr/local/lib** during 'make install'.
2. When rebuilding static secure library all secure library object modules must be removed from 'libshift.a' before inserting newly built ones.
3. All Imakefiles should be modified to allow linking with static secure library.
4. **'/etc/init.d/<server> stop'** doesn't work for several services due to lack of credentials. This is because sysconfig file is used only during startup phase and isn't used during service shutdown. Startup scripts have to be modified for these services.
5. At the moment mapping isn't used in the 'vdqmserv', 'stgdaemon', 'rtcpd', 'tpdaemon'. ID is received but not used. This should be revised.
6. The Security layer currently provides only authentication services, as this is the primary need for CASTOR. However, some encryption, and message integrity could easily be provided if need be, as the main mechanism used will be KRB5 and GSI, both wrapped by the GSSAPI which provides those services.
7. At the moment as the stager and rtcopy have been modified to not use setuid and setgid, all HSM files belong to root! To solve that problem, the various APIs willl have to be modified to allow the servers to specify the identity of the user on behalf of which they are working
8. The errors messages should not be stored in a per-thread variable, but in the security context itself.

We also found some mechanism dependent issues that can create problems when deploying a secure CASTOR:

1. For the GSI mechanism, when the certificate of the certification authority is not found in /etc/grid-security/certificates, an unclear error message saying 'can not receive certificate' is returned.
2. For the GSI mechanism, if the CRL have expired, the error message is not very explicit either.
3. The GSI mechanism forces the certificates to be owned by the same user than the one running the server, otherwise they are rejected. Read access is not sufficient.

4. For Kerberos 5, the servers have to acquire client credentials. based on their keytab. This can be done using 'kinit -k'. However, those tickets expire (depending on site policy) and have to be renewed on a regular basis to avoid problems.

The development of the security layer and of secure CASTOR are still in progress, the issues listed above have to be solved before we can start deploying a full scale installation.

7 Acknowledgments

This work was supported by **CERN** and **Federal Agency for Science and Innovations of the Russian Federation**. We would also like to thank our colleagues Olof Barring, Jean-Damien Durand, Jan Iven, Emil Knezo and Sebastien Ponce for their help and support.

8 Appendix A: The CASTOR Test Certification Authority

8.1 Adding new CA to the GSI

1. Obtain CA certificate, say 'cacert.pem'.
2. ***openssl x509 -in cacert.pem -noout -hash***
   gives you hash value, for example:
   ee72d3ff
3. You have to put cacert.pem to <hash>.0 and move it to /etc/grid-security/certificates
   mv cacert.pem /etc/grid-security/certificates/ee72d3ff.0
4. Create text file with signing policy <hash>.signing_policy
   It must contain three lines, for example:

access_id_CA	X509	'/C=CH/O=CERN/OU=CASTOR/CN=CERN CASTOR CA'
pos_rights	globus	CA:sign
cond_subjects	globus	'"/C=CH/O=CERN/OU=CASTOR/*"'

8.2 Adding CASTOR CA to the GSI

Copy CASTOR CA certificate and signing policy files from /
afs/cern.ch/project/castor/CA to your
/etc/grid-security/certificates directory:

cp /afs/cern.ch/project/castor/CA/ee72d3ff.0 /etc/grid-security/certificates
cp /afs/cern.ch/project/castor/CA/ee72d3ff.signing_policy /etc/grid-security/certificates

8.3 Generating host keys and host certificates

There is a script ***gencert*** in the ***/afs/cern.ch/project/castor/CA*** directory. It accepts two parameters: a type of service and a host name. A type of service must be one of the following: **host**, **central**, **disk** or **tape**. You have to enter pass phrase for the CA private key. The example follows:

[root@lxshare003d CA]# ./gencert tape asis-w8

```
Using configuration from /
afs/cern.ch/project/castor/CA/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
```

```
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'CH'
organizationName      :PRINTABLE:'CERN'
organizationalUnitName:PRINTABLE:'CASTOR'
commonName            :PRINTABLE:'castor-tape/asis-
w8.cern.ch'
Certificate is to be certified until Aug  2 14:13:22 2005
GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
The host key is in /
afs/cern.ch/project/castor/CA/keys/castor-tape-asis-
w8.cern.ch-key.pem, and certificate is in /
afs/cern.ch/project/castor/CA/certificates/castor-tape-
asis-w8.cern.ch-cert.pem
```