Applied cryptography, Spring 2010, bigger project

Consider a system of libraries, books and lenders. Libraries have collections of books that can be taken out and returned by lenders. We model this by considering libraries as servers that the lenders connect to. The lender can search the collection of the library and indicate which books it wants to take out. Similarly, the lender can return books to the library. The communication between libraries and lenders is protected, in order to not expose lenders looking for subversive literature. There is a CA that issues certificates to libraries and lenders (presumably after checking their background).

Set up the CA and build the programs for libraries and lenders. The library has a database of library items (books or magazines) specified in Lab 4 as the `LibraryItem` type. For each item, it stores to which lender the book or magazine has been lent to (or whether the book or magazine is in the library). The library is implemented as a server. After a lender (an interactive program) has made connection with the library, it issues queries of type `LenderQuery`, defined as follows:

```
LenderQuery ::= CHOICE {
     searchQ        [0] SearchQuery,
     borrowQ        [1] BorrowQuery,
     returnQ        [2] ReturnQuery
}
```

Library's answers are of type `LenderAnswer`, defined as follows:

```
LenderAnswer ::= CHOICE {
     searchA        [0] SearchAnswer,
     borrowA        [1] BorrowAnswer,
     returnA        [2] ReturnAnswer,
     failA          [3] UTF8String
}
```

Here the last case contains an error message. For example, it is used when the client wants to borrow a book the library does not have, or wants to return a book it has not borrowed.

The search query is defined as follows

```
SearchQuery ::= SEQUENCE {
     searchFormat       OBJECT IDENTIFIER,
     searchContent      ANY DEFINED BY searchFormat
}

id-fp-s2010 OBJECT IDENTIFIER ::= {1 3 6 1 4 1 3516 15 1 1}
```

The library must accept at least the queries of the following formats:

```
yearQuery OBJECT IDENTIFIER ::= {id-fp-s2010 1}
```

It defines the query format

```
YearQueryFormat ::= INTEGER
```

and the library responding to such query must return all library items (books and magazines) published or issued at the given year.

```
textQuery OBJECT IDENTIFIER ::= {id-fp-s2010 2}
```

It defines the query format

```
TextQueryFormat ::= UTF8String
```

The library responding to such query must return all library items that have something to do with the query string (it is up to you to interpret this).

You are free to define more query types[1]. The library must understand when it is been submitted a query it does not understand.

The answer to a search query contains all library items satisfying the condition:

```
SearchAnswer ::= SEQUENCE OF LibraryItem
```

The borrowing query has the following form

```
BorrowQuery ::= SEQUENCE {
     item           LibraryItem,
     outTime        GeneralizedTime,
     signature      SignedData
}
```

naming the book or magazine to be borrowed, the time when it was borrowed, and for the case that the borrowing will be successful, contains the embedding of a signature on a statement saying that the lender borrowed the book[2]. The `signature` field must contain a signature, the structure of which is defined in the Cryptographic Message Syntax (see RFC 5652). The `encapContentInfo` field of that structure must embed a value with the following structure:

```
BorrowStatement ::= SEQUENCE {
     item           LibraryItem,
     outTime        GeneralizedTime
}
```

Here the fields `item` and `outTime` are the same as in the borrowing query. The object identifier for `BorrowStatement` is

```
id-BorrowStatement OBJECT IDENTIFIER ::= {id-fp-s2010 3}
```

The answer to a successful borrowing query is

```
BorrowAnswer ::= NULL
```

i.e. there is no data to report back.

The return query is

1 Ask the teachers to be given your own branch in the OID tree.

2 In "reality", the signature would only be issued at the same time the physical book is actually handed over to the borrower. This would have been preceded by a conversation to make sure that the wanted book is actually in stock. But as we have no physical books here, let us skip that conversation. We assume that if the book is not in stock, then the signature did not actually exist.

```
ReturnQuery ::= LibraryItem
```

It names the book or magazine to be returned. If the return is successful, the library answers with

```
ReturnAnswer ::= SignedData
```

Its `encapContentInfo` field embeds the following structure

```
ReturnStatement ::= SEQUENCE {
     item           LibraryItem,
     inTime         GeneralizedTime
}
```

with the identifier

```
id-ReturnStatement OBJECT IDENTIFIER ::= {id-fp-s2010 4}
```

Here `item` contains the library item that was received, `inTime` encodes the time it was received. The lender must verify the signature and consider the failure of verification as a failure of returning.

Each of lenders and libraries must have two certificates. One is for establishing a secure connection and the other for signing the borrowing and returning receipts.

One can imagine disputes to arise, where the library says that a lender holds a particular book, but the other denies this. Implement a dispute resolution procedure, where both parties can submit the signatures they hold, and out comes a decision whether the lender must return the book.

We consider **interoperability** as an important feature: one student's lender application must be able to communicate with the other's library implementation. When designing and implementing the programs, please use the course mailing list to discuss potentially incompatibility-inducing choices you are going to make.

In short, you are asked to implement

- A server application: the library. It should work autonomously.
- A client application: the lender. It should have some sort of user interface.
- A dispute resolution application. It can just read all necessary evidence from a local storage and output the decision.

You also must set up a CA and be capable of easily issuing certificates for libraries and lenders. Library and lender programs should have convenient means to indicate which CAs are trusted.