Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment

Anti-Cloning Demonstrator

Authors: Thomas Plos (TU Graz), Manfred Aigner (TU Graz), Antti Ruhanen (Confidex), Trevor Burbridge (BT)

October 2009

This work has been partly funded by the European Commission contract No: IST-2005-033546

This document:

Main aim of the final demonstrator for task 4.3.2 is to familiarize non-technical audience with the principle of secure tag authentication.

This is achieved by selecting a simple RTI (returnable transport item) scenario. RTIs such as pallets, small boxes, containers, and bottles which are used for transporting goods are an integral part of supply chains. Over time, RTIs wear and need to be maintained by their owners. Since this work causes significant costs, owners are interested in only maintaining origin RTIs issued by themselves. This can be achieved by equipping RTIs with security-enabled RFID tags. In that way, origin RTIs can easily be distinguished from copied RTIs by using secure tag authentication.

Disclaimer:

Executive Summary

Main aim of the final demonstrator for task 4.3.2 is to familiarize non-technical audience with the principle of secure tag authentication.

This is achieved by selecting a simple RTI (returnable transport item) scenario. RTIs such as pallets, small boxes, containers, and bottles which are used for transporting goods are an integral part of supply chains. Over time, RTIs wear and need to be maintained by their owners. Since this work causes significant costs, owners are interested in only maintaining origin RTIs issued by themselves. This can be achieved by equipping RTIs with security-enabled RFID tags. In that way, origin RTIs can easily be distinguished from copied RTIs by using secure tag authentication.

The rest of this document is structured as follows. The specification of the final demonstrator is presented in Section 1 followed by an overview of the RTI scenario in Section 2. Implementation details of the final demonstrator are covered by Section 3.

Contents:

1 Specification

The final demonstrator mainly consists of four components:

- a reader application with an easy-to-handle graphical user interface (GUI),
- a UHF reader,
- at least one UHF tag with secure tag-authentication functionality
- a EPCIS repository that provides a standard client query
- a secure web server with a tag database.

Consecutively, more detailed information about the requirements of the individual components is given.

Reader Application with GUI

- should provide easy-to-use GUI
- should be able to communicate with UHF reader (e.g. connect to reader, send ‘Get_Challenge’ and ‘Get_Response’ commands required for secure tag authentication)
- should be able to generate random 128-bit challenges
- should display the ID of the tags in the reader field in an appropriate manner
- should connect to a fixed web server via HTTP(s) and communicate with the server (send queries (ID and 128-bit challenge) to the web server and interpret the responses (128-bit challenge encrypted under the tag key))
- must not be in the possession of the key used for the secure authentication
- should display whether a tag has been authenticated successfully or not

UHF Reader

- using UHF reader from CAEN (e.g. CAEN A828EU compact reader)
- should be able to send ‘Get_Challenge’ and ‘Get_Response’ commands required for secure tag authentication)

UHF Tag

- using security-enabled tag prototypes from CAEN, CONFIDEX, or TUG
- should have integrated a software version of the AES-128 encryption algorithm
- should store a 128-bit key used for secure tag authentication
- should be able to communicate with the UHF reader and correctly interpret ‘Get_Challenge’ and ‘Get_Response’ commands required for secure tag authentication)

Secure Web Server

- should accept a HTTP(s) connection from the reader application
- should contain an SQL database with entries for all tags that are used for the demonstration (containing: ID, corresponding key, and the name of the product the tag is attached to)
- generating new tag entries in the SQL database should be achieved by using an existing SQL-database editing tool)
- should be able to display all tags that are currently available in the SQL database (e.g. webpage that generates a list of all tags (ID, name of the product the tag is attached to)
- should be able to handle queries from the reader application (consisting of the ID and a 128-bit challenge)
- should be able to use the ID to look-up the corresponding key, encrypt the received challenge under this key, and send the encrypted challenge as response to the reader application

The EPCIS-Interface

- The Fosstrack EPCIS repository provides a standard client query GUI that can be used to query and retrieve the authentication results. If another EPCIS repository is used then the manufacturer’s own client query GUI may be used.

2 Overview of the RTI Scenario

The secure tag authentication utilized in the RTI scenario covers 15 main steps which are described below. *Figure 1* provides an overview of the individual steps.

1.) Reader application initiates the reading of the tag ID
2.) UHF reader performs inventory procedure with tag
3.) Tag replies with ID
4.) UHF reader forwards ID to reader application
5.) Reader application sends a 'Get_Challenge' command with the concerning challenge to the UHF reader
6.) UHF reader forwards the 'Get_Challenge' command to the tag
7.) Tag replies with OK
8.) UHF reader forwards message from tag
9.) Reader application sends a query to the web server containing the ID of the tag and the concerning challenge
10.) Web server replies with encrypted challenge (= Server_Response)
11.) Reader application sends 'Get_Response' command to the UHF reader
12.) UHF reader forwards 'Get_Response' command to the tag
13.) Tag answers with encrypted challenge (= Tag_Response)
14.) UHF reader forwards Tag_Response to reader application
15.) Reader application checks whether Server_Response is equal to Tag_Response and displays Authenticated/ not Authenticated

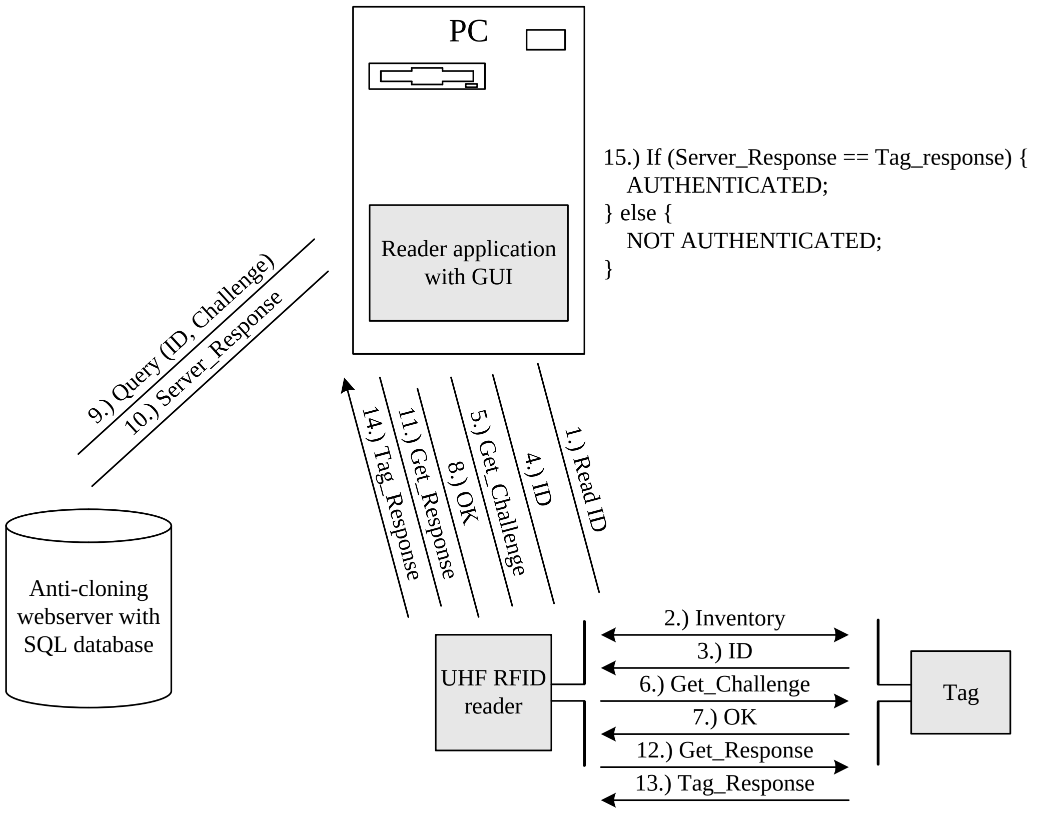


Figure 1 Overview of the individual steps in the RTI scenario.

3 Implementation

The final demonstrator has been implemented according to the requirements specified in Section 1. In the following, details about the implementation of the reader application and the secure web server are provided. The utilized UHF reader and the security-enabled UHF tags are not explicitly described since they have already been covered in deliverable 4.2.1 and deliverable 4.2.2.

Implementation of the Reader Application

The reader application is a simple software with graphical user interface (GUI). It is easy to use and intuitive software for demonstrating tag authentication. Software is used simply by clicking detect and authenticate -button which will execute following steps:

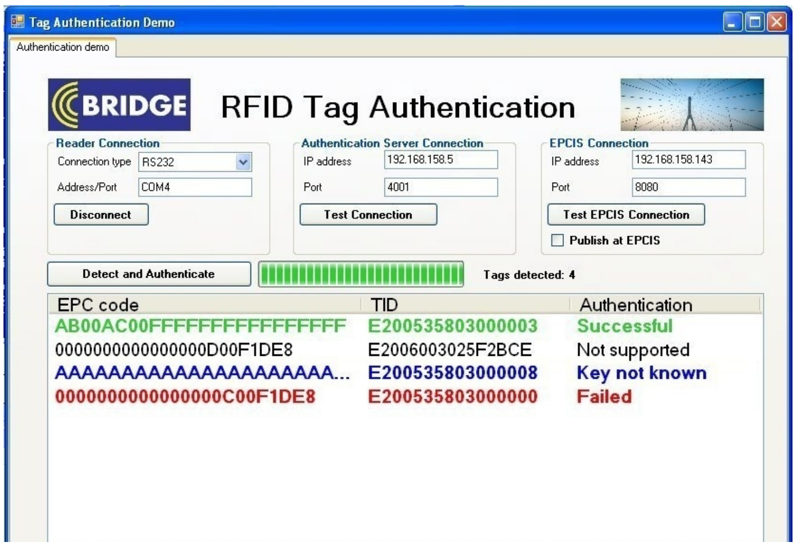
1. Do inventory (find available tags within reader field).
2. Detect if tag supports cryptographic authentication.
3. Challenge the tag with random data and get response.
4. Send same challenge to Secure Web Server together with tag (claimed) identity.
5. Compare responses from tag and from web server. Tag is authentic is responses are equal.
6. Show authentication results in GUI and publish them at EPCIS

It is notable that software itself does not know secret keys. Secrets are controlled in centralized manner (by administrator of Secure Web Server) without limiting the number of clients performing the authentication. This reader application is representing only one of potentially many clients who are performing authentications for secure tags.

Used RFID air interface protocol is totally compatible with EPCglobal gen2. This means that conventional tags (without authentication functionality) are identified as well. There are four possible states for tags which are detected.

1. Authentic: tag is successfully authenticated.
2. Counterfeit: tag supports authentication but authentication results is failed.
3. Unknown: tag supports authentication but authenticity can't be proved since used Secure Web Server does not know this specific tag.
4. Authentication not supported: tag does not support cryptographic authentication.

These states are also illustrated in GUI screenshot below, Figure 2.



RFID Tag Authentication

Reader Connection
Connection type: RS232
Address/Port: COM4
Disconnect

Authentication Server Connection
IP address: 192.168.158.5
Port: 4001
Test Connection

EPCIS Connection
IP address: 192.168.158.143
Port: 8080
Test EPCIS Connection
☐ Publish at EPCIS

Detect and Authenticate
Tags detected: 4

EPC code	TID	Authentication
AB00AC00FFFFFFFFFFFFFFFF	E200535803000003	Successful
0000000000000000D00F1DE8	E2006003025F2BCE	Not supported
AAAAAAAAAAAAAAAAAAAAA...	E200535803000008	Key not known
0000000000000000C00F1DE8	E200535803000000	Failed

Figure 2 Screenshot of the RFID Reader Application.

Software is controlling CAEN RFID-reader which supports special commands for tag authentication. Secure Web Server needs to be available as web service (http-connection) and data transfer to EPCIS is done through standardized EPCIS Capture Interface.

Implementation of the Secure Web Server

The secure web server is a JAVA application that runs locally on a PC using a freely selectable port. Main functionality of the secure web server is the handling of queries from the reader application. Depending on the configuration, the secure web server either accepts HTTP or HTTPS requests.

Overview of the Secure Web Server

The secure web server consists of the executable JAVA archive WebServer.jar containing the code itself and four sub folders: certificate, config, hosting, and log. An overview of the secure web server's file structure is given in Figure .

certificate
Location of the certificate files for secure HTTP communication (HTTPS). By default, the file mySrvKeystore is used that holds a self-signed certificate. The corresponding key-store password is "123456". Any other appropriate certificate file can be used instead.

config
The properties file webserver.properties is responsible for configuring various parameters of the secure web server.

hosting
Contains the files that are hosted by the secure web server.

log
The file webserver.log contains a list of requests handled by the secure web server since its last start.

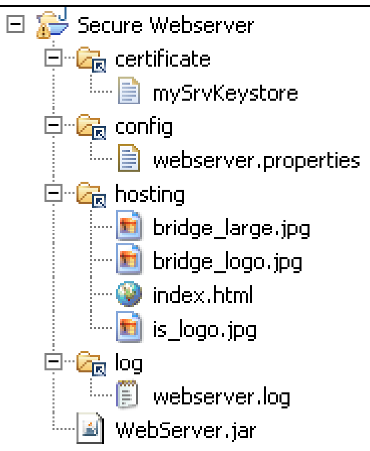
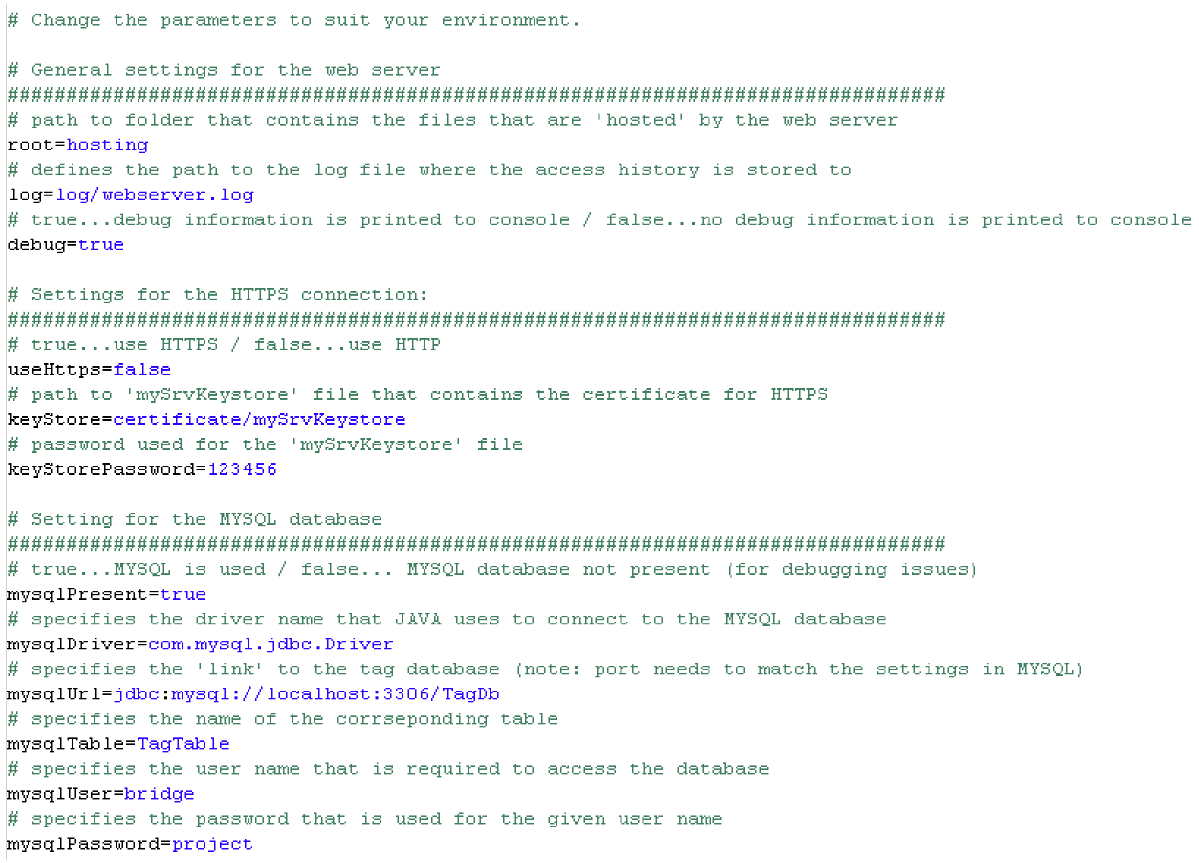


Figure 3 Overview of the secure web server's file structure.

Configuring the Secure Web Server

In order to configure the secure web server, the properties file webserver.properties is used. The file is located in the sub folder config and allows adjusting various parameters concerning the MySQL database, the HTTPS connection, and the secure web server itself. A screenshot of an exemplarily properties file is given in Figure 4.



```
# Change the parameters to suit your environment.

# General settings for the web server
################################################################################
# path to folder that contains the files that are 'hosted' by the web server
root=hosting
# defines the path to the log file where the access history is stored to
log=log/webserver.log
# true...debug information is printed to console / false...no debug information is printed to console
debug=true

# Settings for the HTTPS connection:
################################################################################
# true...use HTTPS / false...use HTTP
useHttps=false
# path to 'mySrvKeystore' file that contains the certificate for HTTPS
keyStore=certificate/mySrvKeystore
# password used for the 'mySrvKeystore' file
keyStorePassword=123456

# Setting for the MYSQL database
################################################################################
# true...MYSQL is used / false... MYSQL database not present (for debugging issues)
mysqlPresent=true
# specifies the driver name that JAVA uses to connect to the MYSQL database
mysqlDriver=com.mysql.jdbc.Driver
# specifies the 'link' to the tag database (note: port needs to match the settings in MYSQL)
mysqlUrl=jdbc:mysql://localhost:3306/TagDb
# specifies the name of the corrseponding table
mysqlTable=TagTable
# specifies the user name that is required to access the database
mysqlUser=bridge
# specifies the password that is used for the given user name
mysqlPassword=project
```

Figure 4 Screenshot of an exemplarily properties file used for configuring the secure web server.

root
: Specifies the relative path to the folder where the hosted files are located.

log
: Relative path and file name to the location where log messages are stored to.

debug
: Defines whether additional debug messages are printed to console (debug = true) or not (debug = false).

useHttps
: Enables (useHttps = true) or disables (useHttps = false) secure communication via HTTPS. Ordinary HTTP communication is deployed if this parameters is set to false. This is especially useful for testing functionality of the secure web server if no HTTPS support is available at client side.

keyStore
: Relative path to location of server certificate.

keyStorePassword
: Password that protects access to the keyStore that contains the server certificate.

mysqlPresent
: Activates (mysql_Present = true) or deactivates (mysql_Present = false) the usage of the MySQL database. This is useful for debugging if no MySQL database is present on the test system.

mysqlDriver
: Specifies the driver that is used to connect to the MySQL database.

mysqlUrl
: Specifies the URL to the location of the MySQL database.

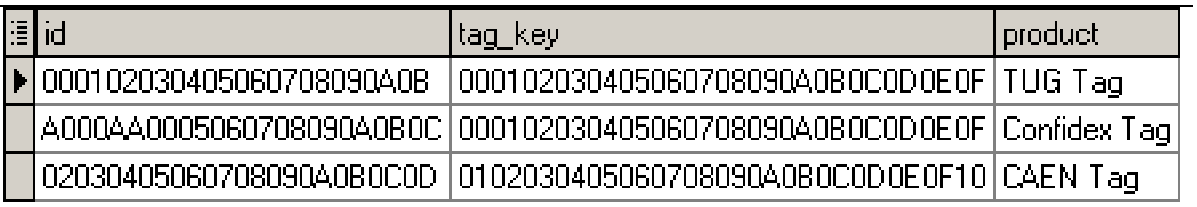
mysqlTable
: Defines the name of the table that is used by the MySQL database.

mysqlUser
: User name for accessing the MySQL database.

mysqlPassword
: Password for accessing the MySQL database.

Starting the Secure Web Server

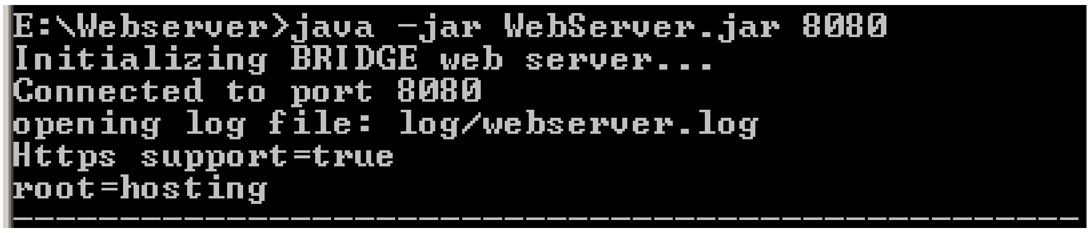
After proper configuration, the secure web server can be started. Running the secure web server requires prior installation of a Java Runtime Environment (JRE). Successful tests of the secure web server were conducted with Java version 1.6.0_11. Moreover, an appropriate MySQL database needs to be present if mysqlPresent is set to true in the properties file (which is necessary for the final demonstrator). The database needs to have three columns: id, tag_key, and product. id is the unique identifier of the corresponding RFID tag (EPC). tag_key is the secret 128-bit key used for secure tag authentication. product is a description of the product to which the tag is attached to. Figure provides an example of a MySQL table that can be used by the secure web server. There exists a variety of freeware tools that allow to display and modify MySQL databases in a rather convenient way (e.g. DreamCoder for MySQL).



id	tag_key	product
000102030405060708090A0B	000102030405060708090A0B0C0D0E0F	TUG Tag
A000AA0005060708090A0B0C	000102030405060708090A0B0C0D0E0F	Confidex Tag
02030405060708090A0B0C0D	0102030405060708090A0B0C0D0E0F10	CAEN Tag

Figure 5 Example of a table used by the secure web server.

Starting the secure web server is accomplished via command line by using the JRE. The port used by the secure web server can be defined via an additional parameter. By default, port 8000 is used if no explicit port number is provided. In the example illustrated in Figure , port 8080 is utilized. After starting the secure web server, it incoming request at the designated ports are handled.



```
E:\Webserver>java -jar WebServer.jar 8080
Initializing BRIDGE web server...
Connected to port 8080
opening log file: log/webserver.log
Https support=true
root=hosting
-------------------------------------------------------
```

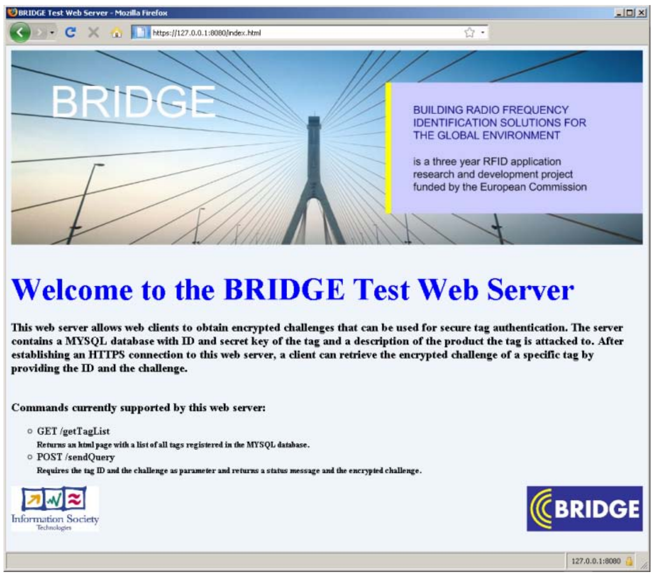
Figure 6 Starting the secure web server via command line.

Testing the Functionality of the Secure Web Server

When the secure web server is running, its functionality can be tested. Three tests are conducted, accessing the welcome page, retrieving an html page listing all tags currently in the MySQL database, and handling queries from the reader application.

a.) Accessing the Welcome Page

Accessing the welcome page of the secure web server is achieved by requesting the file index.html via a standard web browser. A screenshot of the secure web server's response is presented in Figure .



BRIDGE Test Web Server - Mozilla Firefox
https://127.0.0.1:8080/index.html
BRIDGE
Information Society Technologies
BRIDGE
127.0.0.1:8080

BUILDING RADIO FREQUENCY IDENTIFICATION SOLUTIONS FOR THE GLOBAL ENVIRONMENT

is a three year RFID application research and development project funded by the European Commission

Welcome to the BRIDGE Test Web Server

This web server allows web clients to obtain encrypted challenges that can be used for secure tag authentication. The server contains a MYSQL database with ID and secret key of the tag and a description of the product the tag is attacked to. After establishing an HTTPS connection to this web server, a client can retrieve the encrypted challenge of a specific tag by providing the ID and the challenge.

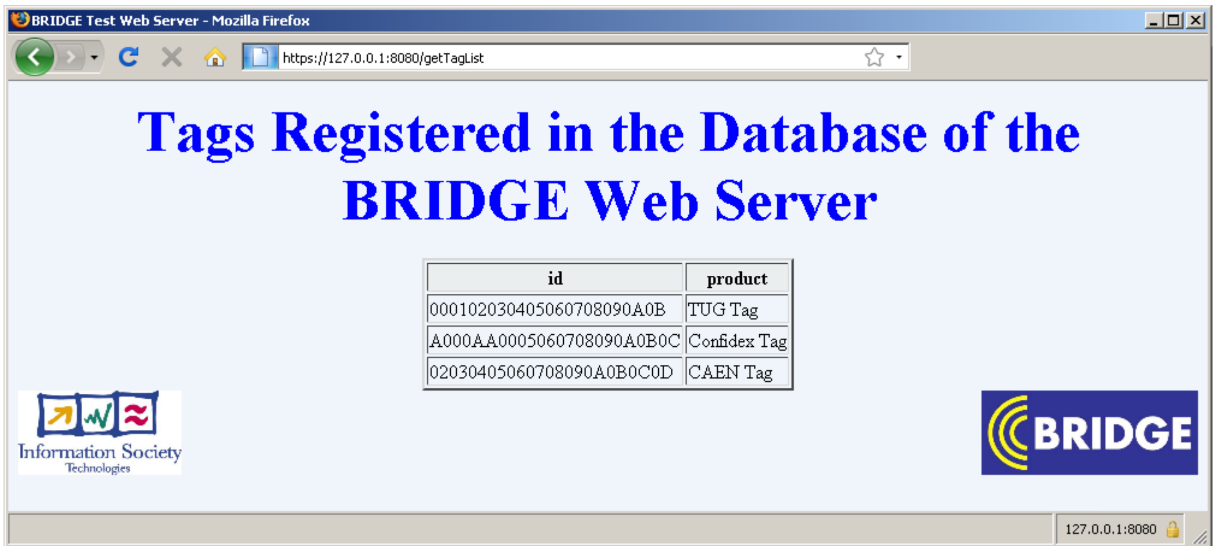
Commands currently supported by this web server:

- GET /getTagList
  Returns an html page with a list of all tags registered in the MYSQL database.
- POST /sendQuery
  Requires the tag ID and the challenge as parameter and returns a status message and the encrypted challenge.

Figure 7 Screenshot of the secure web server's welcom page obtained via a standard web browser.

b.) Accessing the Tag-List Page

Retrieving an html page listing all tags currently in the MySQL database is accomplished by sending the request getTagList to the secure web server. This can be again achieved via a standard web browser. A screenshot of the secure web server's response is presented in Figure .


BRIDGE Test Web Server - Mozilla Firefox

https://127.0.0.1:8080/getTagList

Tags Registered in the Database of the BRIDGE Web Server

id	product
000102030405060708090A0B	TUG Tag
A000AA0005060708090A0B0C	Confidex Tag
02030405060708090A0B0C0D	CAEN Tag

Figure 8 Screenshot of the html page listing all tags currently in the MySQL database.

c.) Handling Queries from the Reader Application

The last test covers the main functionality of the secure web server—handling queries from the reader application. The reader application sends a query containing the tag ID and the challenge via a HTTP POST command to the secure web server as illustrated in Figure 9. The response of the secure web server indicates whether encrypting the challenge was successful (see Figure) or not (see Figure). Moreover, also the challenge encrypted under the secret key of the corresponding tag is replied if the operation was successful.

```
POST /getQuery HTTP/1.0

ID=000102030405060708090A0B&Challenge=000102030405060708090A0B0C0D0E0F
```

Figure 9 Query from the reader application containing tag ID and challenge

```
OK=1&Response=FE53F82A54949856AEB063C1119C8F78
```

Figure 10 Response of the secure web server indicating a successful operation.

```
OK=0&Response=
```

Figure 11 Response of the secure web server indicating an unsuccessful operation.

Integration of an EPCIS

The Reader Application allows the specification or a URL for an EPCIS Capture Interface. This URL can be used to direct authentication results to an EPCIS repository such as the Fosstrack Open Source EPCIS repository.

The Reader Application sends a capture event specified in XML via an HTTP Post to the EPCIS Capture Interface as specified by the EPCglobal EPCIS standard. The EPCglobal EPCIS specification defines a standard Object Event vocabulary which includes the elements: EPC list, timestamps, *action*, *bizStep*, *disposition*, *readPoint*, *bizLocation* and *bizTransaction*.

Authentication information has not been considered within the EPCglobal EPCIS vocabulary but the *bizStep* and *disposition* fields are the most appropriate for such information. It should also be noted that the vocabularies may be extended, and work is currently ongoing to define industry standard parameters for these fields using URNs.

Considering *bizStep*, the most appropriate current parameter is ‘inspecting’ to define a process which checks for physical or documentation defects. Such a parameter may be suitable for electronic or physical authenticity checks (following the failure of electronic authentication). Otherwise an additional ‘authenticating’ *bizStep* value may be added to the standard vocabulary parameter list.

Disposition does not currently include any parameter values suitable for authentication results. In future revisions of the EPCIS vocabulary, such values would need to be defined to allow the common understanding of authentication results within the industry. However, one problem with the use of *disposition* is that authentication results cannot be captured in parallel to other *disposition* values. Thus a preferred option may be to extend the Object Event to include a separate optional authentication parameter. In either case we suggest that authentication results should include both the result of the authentication process (i.e. the subject is authentic, counterfeit or the result is unknown e.g. because the authentication service was down) and the standard authentication process that was applied (EPC/RFID provides a platform for different security measures).

Thus, values may include:

authentic_Crypto	for cryptographic authentication (in reality there are likely to be separate values for different cryptographic schemes)
authentic_TID	for manufacture TID verification
authentic_SSCM	for stochastic track and trace plausibility checks (again several schemes expected in real-life)
counterfeit_crypto	
counterfeit_TID	
counterfeit_SSCM	
unknown_Crypto	
unknown_TID	
unknown_SSCM	

These values may be preceded with a string similar to urn:epcglobal:cbv:disp:auth (depending on the EPCglobal vocabulary evolution) to form an industry standard URN.

Within the BRIDGE authentication demo, the authentication check is based upon AES cryptography and hence only the following values are used:

Authentic_crypto
Counterfeit_crypto
Unknown_crypto

These values match with the results displayed on the GUI of the Reader Application.

The values within the EPCIS repository can be retrieved using the standard EPCIS Query Interface. Using the Fosstrack implementation, the standard client query GUI provided can be used to retrieve the authentication results. In industry deployments such EPCIS repository records may be harvested by a centralised counterfeit monitoring service, or advertised through the use of a Discovery Service to other organisations within the supply chain.

Abbreviations:

AES	Advanced Encryption Standard
EPC	Electronic Product Code
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ID	Identification (number)
JRE	Java Runtime Environment
RFID	Radio frequency Identification
RTI	Returnable Transport Item
SQL	Structured Query Language
UHF	Ultra High Frequency