EOSS CRYPTO TOKEN

Electronic Optical Sealing System
Crypto Token

JOPAG/10.02-DOC-

Joint Programme

on the

Technical Development and Further Improvement of IAEA Safeguards

between the Government of the Federal Republic of Germany and the International Atomic Energy Agency

TASK D.27/E994

Crypto Token
Electronic Optical Sealing System
EOSS
Instructions and Reference

Forschungszentrum Jülich GmbH
D-52425 Jülich

July 2007

Revision History

Edition	Issued	Status	Token Firmware Version
1.00	July 2007	Draft	1.0

Table of Contents

1 Introduction

The Electronic Optical Sealing System (EOSS) uses cryptographic methods for access authorization and data encryption and authentication. The cryptography bases on numeric keys that must be kept secret from unauthorized persons.

The crypto token is a secure and easy to use container for EOSS keys. The token stores the keys and performs the sensitive cryptographic operations. Stored keys cannot be retrieved from the crypto token. In other words, the token is a combination of crypto engine and write-only key storage device.

The EOSS crypto token provides the option to protect stored keys with a password. If a password is set, then the keys will be stored in an encrypted format. The password protection makes it even harder for adversaries to extract keys from an illegally obtained crypto token. Moreover, it prevents unauthorized persons from using such a token to read data from EOSS seals.

The EOSS crypto token is an USB device. It has the same form factor as the popular USB memory sticks. One token provides storage capacity for the keys of approximately 700 EOSS seals.

The tokens are delivered with drivers and application programs for Microsoft Windows. The software allows other software modules, for example the EOSS Reader, to use the services of inserted crypto tokens.

The application software also provides the possibility to access EOSS keys from a database system. This option is used in the headquarters to load required keys into the crypto tokens.

2 Software Installation

2.1 USB Drivers

The EOSS crypto token requires universal serial bus (USB) and virtual comport (VCP) drivers from Future Technology Devices International Ltd. (FTDI).

A combined driver setup executable is available on the EOSS Reader installation CD. More recent versions of the driver may be obtained from the FTDI website (http://www.ftdichip.com).

To install the drivers, run the setup executable and follow the instructions.

2.2 Application Software

The EOSS crypto token application software consists of an executable file TokCom.exe. The file is automatically installed along with the EOSS Reader software.

The EOSS Reader setup utility copies the file into the destination folder and registers TokCom as a component object model (COM) server. Moreover, it places a link into the start-up section of the start menu. The link automatically starts TokCom when a user logs on. To start TokCom after the installation, log off and re-log on or restart the system.

3 Using a Crypto Token

Both the EOSS seal and the crypto token were designed to allow all cryptographic operations to be performed fully automatically. Especially, it is not necessary to "manually" type or select cryptographic keys.

To use a crypto token, just insert it into an USB port of your PC or notebook. If asked, provide the correct password to unlock the token. The token is now ready to be used by application programs as for instance the EOSS Reader.

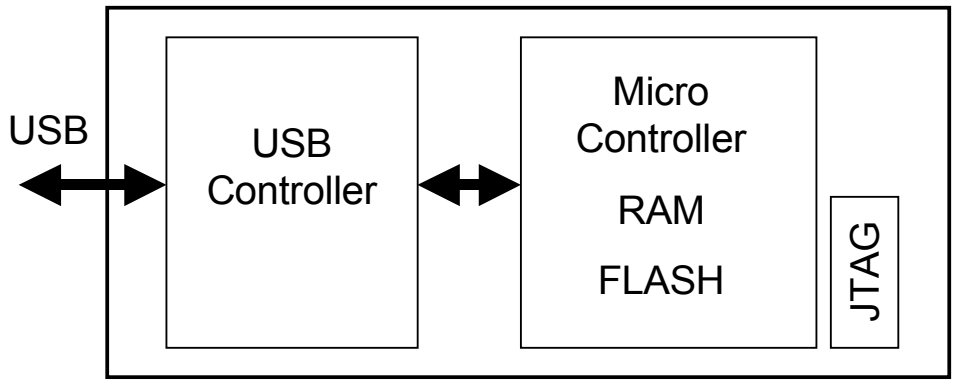
The token is equipped with a light emitting diode (LED) that flashes whenever the token is accessed by the host system. This happens in regular intervals in order to check that the token is still inserted, and, of course, whenever the token is performing cryptographic operations. If the LED does no flash, then it is most likely that drivers and/or application software were not properly installed.

You can check the contents of the crypto token at any time. Just click the token icon in the taskbar notification area. A dialog box will be opened that contains a list of all key-sets that are stored in the token.

4 Crypto Token Internals

4.1 Hardware

The crypto token incorporates an USB controller and a micro controller. The first is dedicated to the communication with the host system. The latter performs the cryptographic operations and stores the EOSS keys in its internal Flash (non-volatile) memory.



All resources of the micro controller are accessible through its JTAG (Joint Test Action Group) debug and programming interface. The Flash memory can be written and read (firmware and keys)! However, the JTAG interface is equipped with an internal security fuse. Blowing this fuse, permanently disables the JTAG interface.

The micro controller is also assessable through its ROM resident bootstrap loader (BSL). The BSL cannot be disabled. However, it is protected through a 256-bit password.

Upon delivery, the JTAG port is enabled and the BSL password is set to a default value. More information on how to change the BSL password and blow the JTAG fuse is available on request.

4.2 Firmware

4.2.1 Cryptosystem

EOSS uses the data encryption algorithm that is described in the data encryption standard (DES). The seal always uses DES-EDE mode, also known as 112-bit TDES. The EOSS crypto token supports TDES encryption and decryption. The data block length is 64 bit.

The token does not support the hash algorithm that is required for authentication (DES cipher block chaining). The hash algorithm uses a constant key, which is not a secret. Therefore, hashing can be securely performed on the host system.

The password protection of the EOSS crypto token uses 56-bit DES encryption (electronic code book (ECB) mode).

4.2.2 Key Storage

The EOSS seal uses four different DES-EDE keys. Each key is used for a particular operation. Three of the keys are used for the challenge-response communication protocol (authorization, identification and encryption key). These three keys are required to communicate with a seal. The fourth key, called the authentication key, is of special importance. It enables the owner to verify the genuineness of the seal data (e.g. openings and closings of the seal wire).

The four keys together form a key-set. The key-set is the basic storage item for the crypto token. Each key-set has a unique number, which is composed of the serial number of the seal that generated the keys and an additional consecutive number. Moreover, the key-set contains the date and time of the key generation. This key-set information is used to select a particular key-set on the crypto token.

As already mentioned, the authentication key needs special attention. It's in the nature of symmetric cryptosystems like DES, that authentication keys could be abused to fake, falsify or delete data items. Hence, it is common practise to not allow the seal's authentication key to leave the headquarters (except, of course, inside the seal itself). The EOSS crypto token supports such a policy by treating the authentication key as optional. The authority that manages the crypto tokens can individually decide if authentication keys are to be stored on a token or not.

A crypto token provides storage capacity for about 700 key-sets. The token's command interface allows the host system to read the key-set information (key-set number and key-set date/time) in order create a list which key-sets are available on the token. It is also possible to determine if authentication keys are available. It is not possible the access the stored keys themselves.

Key-sets are stored individually, one after the other. New key-sets can be added at any later time. However, it is not possible to individually erase certain key-sets. Issuing an erase command to the token erases all key-sets. The erase operation is cryptographically secure, that is, all sensitive data will be overwritten with 1-bits.

As long as an token is empty after an erase, it is possible to set a password. This password will be used to encrypt all subsequently stored key-sets. If a token with a set password is inserted to an USB port, it starts operation in a locked state. In this state, most functions of the token are disabled. To unlock a token, the correct password must be provided to the token. When a particular key is required for a cryptographic operation, it is temporarily decrypted into volatile memory.

An unlocked token remains unlocked as long as it is powered through the USB connector.

4.2.3 Command Reference

The purpose of this chapter is to give an overview on the communication between the crypto token and the host system. The provided information can be used to evaluate the security of the token. However, it's not a programming guide. For information on how to use the services of the EOSS crypto token on Microsoft Windows platforms, please refer to chapter 5 which documents the COM interface of the crypto token server.

The communication to the crypto token bases on a request-reply scheme, also known as ACK/NACK protocol (acknowledge/not-acknowledge). The host system sends a request and waits for the corresponding reply from the token. A request packet consist of a command code and optional data. A reply packet consists of a status code and optional data. The returned status code informs the host system on the token's locking state and whether an error occurred.

The following commands are available:

Command	**Request-data**	**Reply-data**	**Description**
Ping	▪ none	▪ S/N	Sent out periodically in order to detect inserted tokens. The reply data contains the serial number of the crypto token. Available in locked and unlocked state.
Unlock	▪ Password ▪ Verification-code	▪ none	Sent to unlock a password-protected crypto token. The 64-bit password is transmitted as plain text. The verification code allows the token to detect invalid passwords.
Info	▪ Key-set index	▪ Capacity ▪ Used ▪ Key-set info	Sent iteratively in order to create a list of key-sets that are stored on the token. The first request uses a key-set index of zero. The token answers with the storage capacity and how much storage is already used. The key-set info contains the key-set number, date and time of the key-set generation and flags which of the four TDES keys are contained in the key-set (e.g. authentication key available or not). The command is repeated until Index is equal to Used. Only available in unlocked state.
Select	▪ Key-set index	▪ None	Selects a key-set to be used for subsequent encryption and/or decryption operations. Only available in unlocked state.
K1-Decrypt	▪ Cipher text	▪ Plain text	Decrypts the 64-bit input block by means of the first key of the selected

			key-set. Only available in unlocked state.
K1-Encrypt	▪ Plain text	▪ Cipher text	Encrypts the 64-bit input block by means of the first key of the selected key-set. Only available in unlocked state.
K2-Decrypt	▪ Cipher text	▪ Plain text	Decrypts the 64-bit input block by means of the second key of the selected key-set. Only available in unlocked state.
K2-Encrypt	▪ Plain text	▪ Cipher text	Encrypts the 64-bit input block by means of the second key of the selected key-set. Only available in unlocked state.
K3-Decrypt	▪ Cipher text	▪ Plain text	Decrypts the 64-bit input block by means of the third key of the selected key-set. Only available in unlocked state.
K3-Encrypt	▪ Plain text	▪ Cipher text	Encrypts the 64-bit input block by means of the third key of the selected key-set. Only available in unlocked state.
K4-Decrypt	▪ Plain text	▪ Cipher text	Decrypts the 64-bit input block by means of the fourth key of the selected key-set. Only available in unlocked state.
K4-Encrypt	▪ Cipher text	▪ Plain text	Encrypts the 64-bit input block by means of the fourth key of the selected key-set. Only available in unlocked state.
Lock	▪ Password ▪ Verification-code	▪ none	Protects a token by means of a 64-bit password. Only available if the token is empty.
Erase	▪ Verification-code	▪ none	Erases all key-sets on the token. The verification code must be the numeric representation of the character string “ERASEALL”.

5 Application Programming Interface

5.1 Crypto Token COM Server

The crypto token COM server is a WIN32 executable program (TokCom.exe). It is usually launched automatically through a link in the startup section of the start menu. The program permanently monitors the ports of the system. If it detects an inserted crypto token, it first checks the locking state of the token. If the token is password protected, a dialog box is opened that allows the user to type the password. If the provided password is correct, the token will be unlocked. TokCom then interrogates the token for its available key-sets. The result is stored in an internal list. Finally, the program places an icon into the taskbar notification area. The token is then ready to be used.

TokCom.exe continues to monitor the ports even after a crypto token was detected. This way, it is possible to simultaneously use multiple tokens. Moreover, it allows the program the detect the removal of a token.

The crypto token COM server provides both the user interface and the application programming interface. The user interface consists of a dialog box that can opened through the taskbar icon. The programming interface is provided through a COM object with four methods: Hash(), Encrypt(), Decrypt() and Store(). The first three methods can be used to communicate with a seal or process seal data. The latter is used to store a newly generated key-set in a database.

All interfaces use Visual Basic string types (BSTR) to transport data across process boundaries. TokCom expects the data in the BSTRs to be in binary format. Seal data has to be passed along with length information. The length must be specified as the number of DES blocks (8 bytes units). All functions return the success status as standard HRESULT values.

The Hash() method provides the DES-CBC algorithm that is required to generate authentication codes. The Hash algorithm uses a fixed key. Hence no key information is required.

The Encrypt() and Decrypt() methods require the key-set number, key-set time and the particular key number (authorization, identification, encryption or authentication key). TokCom searches the specified key in the key-set lists of the inserted tokens. If a suitable token is available, data blocks are subsequently passed to the token. The encrypted or decrypted data is returned to the application.

The crypto token server usually refers all cryptographic tasks to the inserted tokens. However, the server is also capable to perform crypto operations without a token. This mode requires the key-sets to be available through a database connection. The DES and DES-EDE algorithms are provided by the MS Enhanced Cryptographic Service Provider through the Windows CryptoAPI.

5.2 Interface Definition Reference

The following code is from the Interface Definition Language (IDL) file that defines the COM interface of the TokCom crypto token server:

```
import "oaidl.idl";
import "ocidl.idl";

[
	object,
	uuid(6BEC4536-CFF0-4B30-90A2-DF491BFBD7E8),
	dual,
	nonextensible,
	helpstring("IEossCrypto Interface"),
	pointer_default(unique)
]
interface IEossCrypto : IDispatch{
	[id(1), helpstring("method Hash")]
	HRESULT Hash([in] BSTR* pData, LONG Count,
		[in, out] BSTR * pHash);
	[id(2), helpstring("method Encrypt")]
	HRESULT Encrypt(LONG KeysetNumber, LONG KeysetTime,
		LONG KeysetKey, [in,out] BSTR* pData, LONG Count);
	[id(3), helpstring("method Decrypt")]
	HRESULT Decrypt(LONG KeysetNumber, LONG KeysetTime,
		LONG KeysetKey, [in,out] BSTR* pData, LONG Count);
	[id(4), helpstring("method Store")]
		HRESULT Store(LONG KeysetNumber, LONG KeysetTime,
		[in] BSTR AuthenticationKey,
		[in] BSTR AuthorizationKey,
		[in] BSTR IdentificationKey,
		[in] BSTR EncryptionKey,
		[in] BSTR VACOSSKey);
};

[
	uuid(D4F809F4-9CC4-4938-B85E-0476A3F1B3DE),
	version(1.0),
	helpstring("TokCom 1.0 Type Library")
]
library TokComLib
{
	importlib("stdole2.tlb");
	[
		uuid(A410FA1F-3C6B-4881-818A-E34775311858),
		helpstring("EossCrypto Class")
	]
	coclass EossCrypto
	{
		[default] interface IEossCrypto;
	};

};
```

5.3 Sample Reader Application

A sample reader application is available that demonstrates how an EOSS seal can be read with support of the TokCom crypto token server. The sample reader was written in C++. It runs on Windows XP.

The source code is available on request.

6 Database Interface

6.1 Enabling Database Access

The TokCom crypto token server is capable to access a data source through ODBC (open database connectivity).

Database access can be enabled by performing the following two steps:

1. Create an ODBC data source. In the Windows Control Panel, go to Administration – Data Sources (ODBC)! Click Add and select a driver! For a mdb-file, select the Microsoft Access Driver! Click Finish! The next steps depend on the selected driver. With MS Access files, you got to specify a Data Source Name (e.g. "EOSS") and select the mdb-file.
2. Provide the ODBC data source name as an command line parameter to the crypto token server. The server is started by link in the Startup section of the Start menu. To provide parameters, right-click the link and chose Properties. Under Target, add the name of the ODBC data source. You may also add an user name and a password. Caution, if the path to TokCom.exe contains blanks, then it is recommended to enclose the path in quotation marks. Restart the server by logging off and re-logon.

After performing these steps, you may test the connection. Insert a token and click the icon in the taskbar notification area! Click the Add button! If the Add Key-sets to Token dialog appears, the database connection is available.

6.2 SQL Queries Reference

This chapter gives an overview on the SQL queries that the crypto token server uses to communicate with the ODBC data source. The queries partly contain placeholders for the actual parameters. The `%u` and `%s` placeholders stand for numeric and string literals to be inserted by means of the `swprintf()` C++ runtime library function. The `?` signs stand for ODBC parameters that are used along with "prepared" SQL statements.

The crypto token server uses the following SQL statement in order to add a key-set to the Keys table of the database:

```
INSERT INTO Keys (SealID, KeySetNumber, KeysetTime, Authentication,
      Authorization, Identification, Encryption, Vacoss)
VALUES (%u, %u, '%s', '%s', '%s', '%s', '%s', '%s')
```

The following SQL statement is used to retrieve a certain key-set from the Keys table:

```
SELECT Authentication, Authorization, Identification, Encryption
FROM Keys
```

```
WHERE KeysetNumber = %u AND KeysetTime = {ts '%s'}
```

The following SQL statement is used to obtain the operating divisions from the OpDivs table:

```
SELECT OpDivID, DivisionName
FROM OpDivs
```

The following SQL statement is used to obtain the facilities that belong to a selected operating division:

```
SELECT Facilities.FacilityID, FacilityDesignator, FacilityName
FROM Facilities INNER JOIN Requests
     ON Facilities.FacilityID = Requests.FacilityID
WHERE OpDivID = %u
```

The following SQL statement is used to obtain the seals that currently belong to an operating division or facility. The statement is complemented at runtime with a column name and a key. If a division is selected, the column name must be `OpDivID` and the key has to be the ID of the selected division. Otherwise, the column name must be `FacilityID` and the key specifies the selected facility. The DateRequest field of the Request table is used to restrict the results to only those seals, that are ***currently*** assigned to the selected division or facility.

```
SELECT JoinedTable.SealID
FROM  [SELECT * FROM SealRequests INNER JOIN Requests
       ON SealRequests.RequestID = Requests.RequestID].
      AS JoinedTable
      INNER JOIN
      [SELECT SealRequests.SealID,
                  Max(Requests.DateRequest) AS MaxDateRequest
       FROM (SealRequests INNER JOIN Requests
            ON SealRequests.RequestID = Requests.RequestID)
            GROUP BY SealRequests.SealID].
      AS SubQuery
      ON (JoinedTable.SealID = SubQuery.SealID)
            AND (JoinedTable.DateRequest = SubQuery.MaxDateRequest)
WHERE ((JoinedTable.%s) = %u)
```

The following SQL statement is used to get a list of all seals in the database:

```
SELECT SealID
FROM Seals
```

The following SQL statement is used to obtain a sorted list of the key-sets that belong to a certain seal:

```
SELECT KeyID, KeysetNumber, KeysetTime
FROM Keys
WHERE SealID = ?
ORDER BY KeysetTime DESC
```

The following SQL statement is used to retrieve a certain key-set from the Keys table. In contrast to the statement above, the KeyID key is used here.

```
SELECT KeySetNumber, KeysetTime, Authentication, Authorization,
Identification, Encryption
FROM Keys
WHERE KeyID = ?
```

This statement translates a seal serial number to the corresponding SealID.

```
SELECT SealID
FROM Seals
WHERE SerialNumber = ?
```

This statement translates a SealID to the corresponding seal serial number.

```
SELECT SerialNumber
FROM Seals
WHERE SealID = ?
```

This statement creates a new entry in the Seals table. The statement is used when a key-set is about to be stored in the Keys table and the seal that generated the key-set is not yet registered in the Seals table.

```
INSERT INTO Seals (SerialNumber, InventoryNumber, OpStatus)
VALUES (?, 'XXXX/XXX', 'XXX')
```