Functional Specification Swissbit TSE SMAERS Firmware

Version 1.2.11

Swissbit

Dec 10, 2019

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Overview

The Swissbit TSE (*Technische Sicherheitseinrichtung*) provides a way for developers of cash registers (host system) to comply with the upcoming German fiscal laws becoming effective in 2020. The TSE comes in three form factors: micro SD, SD, and USB and can be integrated into an existing cash register system by the use of basic file operations. The Swissbit TSE contains a certified software (the TOE or the Swissbit TSE SMAERS Firmware). This specification describes the use of the TSE at its external interfaces, independent of the form factor.

The TSE described in this specification complies with [BSI-TR-03153] [BSI-TR-03151] and [PP-SMAERS]. It further utilizes the services of a certified security module that is compliant to [PPC-CSP-TS-Au].

The TSE is equipped with a digital certificate that can be used for registration at german tax authorities. Please refer to [AGD] for more information on the process to register the TSE with the german tax authorities.

The TSE covers all the functionality that is required by [BSI-TR-03153] [BSI-TR-03151] and [PP-SMAERS] so that the host does not have to care about secure storage or cryptographic processing. Specifically, the TSE provides a way to securely store transactions in a consecutive and unmodifiable order combined with a digital signature for proof of origin. In case the TSE storage gets filled up, all transactions can be backed up, deleted from the TSE and new transactions can be recorded.

Note: Swissbit also develops a library in ANSI-C as recommended in [BSI-TR-03151] (which is not part of this certification process) that wraps the communication to the TSE and provides an easy way to use the interface for accessing the device. This software library is not described in this specification.

CHAPTER

TWO

PHYSICAL INTERFACE

The TSE is available in three different configurations:

- as a USB token
- as an SD card
- as a micro SD card

Figure Fig. 2.1. shows the physical boundaries of the TSE for the different configurations.

Fig. 2.1: Overview over all configurations of the TSE

The physical interface of the the USB token is implemented in accordance with [USB-Spec] while the SD card configurations are implemented in accordance with [SD-Spec].

Note: The physical interface of the TSE is considered being a TSFI in the context of the Common Criteria evaluation.

CHAPTER

THREE

FILE BASED INTERFACE

The TOE exposes all its services via a file based interface. To do so, the TOE provides a file system that is formatted as FAT32 (please refer to [FAT32] for a complete specification). Some of the space of the file system is pre-occupied to reserve space for the TOE transactions; this space is referred to as the *TSE Store*. The remaining space is freely usable like on a standard storage device.

The *TSE Store* is protected against modification and deletion and can only be read after the host system has been authorized. Please refer to Section 4.1 for more information. Also, the partitioning and file system of the TOE is protected and cannot be changed during the life cycle of the TOE (please refer to Section 5.2 for details).

The overall architecture of the TSE is shown in Fig. 3.1.

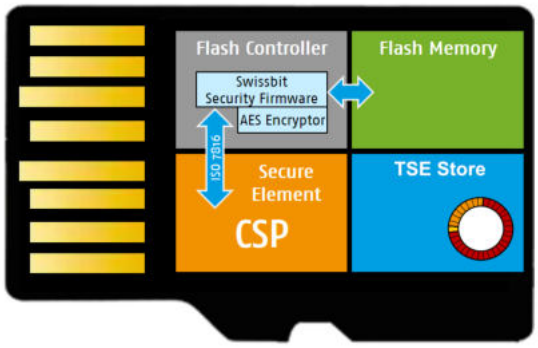

Fig. 3.1: TSE Architecture

3.1 Working principle of the TOE

The overall working principle of the TOE bases on a set of files that are visible on the file system of the TOE after it has been connected to a host system:

Fig. 3.2: TOE Special Files

TSE_COMM.DAT is a communication file, into which the host simply writes specially formatted data in order to execute commands of the TOE. The response of the TOE can be retrieved by reading from the same file again. Please refer to *Commands* for an overview over all commands that are offered by the TOE.

TSE_INFO.DAT contains status information and is read only. Please refer to *TSE Status* for more information.

The *TSE_TAR.xxx* files contain the transaction history. Because of file system limitations, the *TSE Store* is split across multiple files each containing at most 1 GB of data.

Note: The TSE provides a set of commands to record data in form of transactions, which are then saved in the area reserved for the *TSE Store*. Please refer to Section 4 for the relevant commands.

In case any of the special files gets unintentionally deleted, the file and its contents will be automatically restored after the next power cycle.

3.2 TSE Status

Status information about the TOE can be retrieved by reading the file *TSE_INFO.DAT* which has a fixed size of 512 bytes.

The content of this file has the following structure:

Table 3.1: Data structure of *TSE_INFO.DAT*

Field	Size	Offset	Comment
Customization Identifier	4	0	Customization Identifier
Firmware Type	4	4	Either "RLS" for production ready FW, "DEV" for development FW that can be used by ECR vendors, or "TST" for internal test revisions. The string is null-terminated, i.e. the last byte is set to 0.
Firmware ID	4	8	If *Firmware Type* is "TST", then this contains an internal id that identifies the test build. Otherwise, this field is set to 0.
RFU (reserved for future use)	8	12	RFU
TSE Capacity	4	20	Size of *TSE Store* in sectors. Big endian.
TSE Current Size	4	24	Used size of *TSE Store*. Only valid if the *TSE Store* is readable (see Section 3.3), otherwise 0. Big endian.
TSE Security	1	28	Bit 0: xxxxxxx1 Valid Time Set Bit 1: xxxxxx1x Self Test Passed Bit 2: xxxxx1xx CTSS Interface Active (requires successful self test) Bit 3: xxxx1xxx Export allowed if CSP test fails (see Section 4.3.9) Bit 4: xxx1xxxx RFU Bit 5: xx1xxxxx RFU Bit 6: x1xxxxxx RFU Bit 7: 1xxxxxxx RFU
TSE Initialization State	1	29	0: Uninitialized 1: Initialized 2: Decommissioned
Data Import Initialized	1	30	0: No Data Import Initialized 1: Data Import Initialized, not yet finalized (see Section 4.5.1)
Special features	1	31	Bit 0: xxxxxxx1 Android Support. Always 1. Bit 1: xxxxxx1x RFU Bit 2: xxxxx1xx RFU Bit 3: xxxx1xxx RFU

Continued on next page

Table 3.1 – continued from previous page

Field	Size	Offset	Comment
Initial Password States	1	32	Bit 0: xxxxxxx1 Initial PUK changed Bit 1: xxxxxx1x Initial Admin PIN changed Bit 2: xxxxx1xx Initial TimeAdmin PIN changed
RFU	3	33	RFU
Time Until Next Selftest	4	36	Timeout in seconds after which the state *selfTestRun* will automatically be made inactive. Please see Section 4.2.1 for details.
Started Transactions	4	40	Number of transactions that have not been finished, yet. If this equals *Max Started Transactions*, no new transactions can be started until at least one transaction has been finished. Only valid if the *TSE Store* is readable (see Section 3.3), otherwise 0. Big Endian.
Max Started Transactions	4	44	Maximum number of started transactions, i.e. amount of transactions that can be started in parallel. In the current revision, this is 512. Big Endian.
Created Signatures	4	48	Amount of signatures that have been created with this TSE. Please note that this value might exceed *Max Signatures*, since *Max Signatures* is only a soft-cap and it might be possible to actually create more signatures. Big Endian.
Max Signatures	4	52	Maximum amount of signatures that can be created with this TSE. Big Endian.
Registered Clients	4	56	Number of currently registered clients (see Section 4.2.2). Big Endian.
Max Registered Clients	4	60	Maximum number of clients that can be registered. In the current revision, this is 100. Big Endian.
Certificate Expiration Date	8	64	Timestamp (as seconds since Unix Epoch) after which the certificate of this TSE will be invalid. The timestamp will be interpreted as an unsigned number, which means only dates after 1970 are supported. Big Endian.
TAR Export Size	8	72	Size of the whole *TSE Store* in bytes, if exported (see Section 3.4). Only valid if the *TSE Store* is readable (see Section 3.3), otherwise 0. Big Endian.
TSE Hardware Version	4	80	2 Byte Major Version (Big Endian) 1 Byte Minor Version 1 Byte Patch Version
TSE Software Version	4	84	2 Byte Major Version (Big Endian) 1 Byte Minor Version 1 Byte Patch Version

Continued on next page

Table 3.1 – continued from previous page

Field	Size	Offset	Comment
TSE Form Factor	4	88	Either “uSD”, “SD”, or “USB” as null-terminated string. The remaining bytes are filled with zeros.
RFU	2	92	RFU
Max Time Synchronization Delay	4	94	Interval (in seconds) after which command *Update Time* must be sent. Big endian.
MAX_UPDATE_DELAY	4	98	Interval (in seconds) after which a started transaction must have received an update in case new data is available on the cash register. This is currently set to 45 seconds according to *MAX_UPDATE_DELAY* from [BSI-TR-03116-5]. Big endian.
Last Header Block Index	4	102	Sector offset of last TSE entry. This allows to read the *TSE Store* starting from the end. Only valid if the *TSE Store* is readable (see Section 3.3), otherwise 0. Big Endian.
TSE Public Key Length	1	106	Usable length of *TSE Public Key*. Maximum length is 100 Bytes.
TSE Public Key	100	107	Public key that belongs to the private key generating signatures, formatted according to [BSI-TR-03111] 3.2.1 Uncompressed Encoding. Bytes after *TSE Public Key Length* are filled with `0x0` and can be discarded. This key can be used to verify all signatures created by the TSE.
RFU	49	207	RFU
TSE Serial Number	32	256	Raw SHA-256 hash over the public key that belongs to the private key generating signatures. This can be used as TSE unique ID.
TSE Description	128	288	NULL terminated ASCII string containing a short description of the TSE.
RFU	96	416	RFU

Note: Please note that the combination of the fields TSE Hardware Version, TSE Software Version and TSE Form Factor is required to uniquely identify the version of the TSE.

Certificate Expiration Date

After the signing certificate of the TSE has expired, new Log Messages can be signed and will be stored to the *TSE Store*, but the command that leads to the signature creation will fail with `0x100A`: Certificate expired. Please note that the comparison for the expiration is done against the timestamp as found in the generated signature and thus after the signature has been created, not before. That also means that for command *Update Time*, the new timestamp will serve as reference value, not the old one.

Already stored Log Messages can still be exported, but the regular access control restrictions still apply (see Section 3.3).

3.3 Reading TSE Data

All transaction logs are stored in the files *TSE_TAR.xxx* and can be read from there.

The *TSE Store* is filled with *TSE Current Size* number of 512 byte blocks. The maximum number of TSE blocks can be up to *TSE Capacity*.

The *TSE Store* is a [BSI-TR-03153] compliant TAR archive containing each signed Log Message as a separate file. Each file object included in the tar archive is preceded by a 512-byte header record. The file data is written unaltered directly following the header, but is rounded up with zeros to a multiple of 512 bytes. The content of the archived files complies with the ASN.1 structure defined by [BSI-TR-03151]. For reference, the header fields used by this implementation are given in Table 3.2.

The data in the *TSE Store* is only readable if the *CTSSInterfaceState* is active, otherwise blocks filled with 0x00 will be returned.

There is however, one exception from this access control rule: If the self test of the TSE fails due to a malfunction of the builtin security module, the *CTSSInterfaceState* will not become active. If in this situation the *TSE Security* setting has bit 3 set, the TOE will allow to read the *TSE Store* if the *CTSSInterfaceState* has not been actively disabled before (see Section 4.3.2). This allows data recovery in cases where the security module fails. It should be noted however that this behavior must be activated in advance (see Section 4.3.8).

To summarize, reading the *TSE Store* is allowed if either the *CTSSInterfaceState* state is active or all of the following conditions are met

- the *TSE Security* setting has bit 3 set by the use of the command described in Section 4.3.8
- the *CTSSInterfaceState* has NOT been disabled before by the use of the command described in Section 4.3.2
- all self tests of the the TSE have passed except those related to the CSP.

In all other cases, the TOE will only return zeros when reading the *TSE Store*.

Table 3.2: Data model of transaction data in *TSE Store*

Field	Size	Offset	Comment
name	100	0	File name as NULL terminated ASCII string.
RFU	24	100	
size	12	124	Size of the file following the header. The size is represented by a decimal number encoded as an octal number in ASCII.
mtime	12	136	Timestamp when this Log Message has been created. It is the ASCII representation of the octal value represented as an integer number of seconds since January 1, 1970, 00:00 Coordinated Universal Time.
chksum	8	148	The chksum field is the ASCII representation of the octal value of the simple sum of all bytes in the header block. Each 8-bit byte in the header is added to an unsigned integer, initialized to zero, the precision of which shall be no less than seventeen bits. When calculating the checksum, the chksum field is treated as if it were all blanks.
typeflag	1	156	Always ASCII character ‘0’.
RFU	100	157	
magic	6	257	Always the NULL terminated ASCII string ‘ustar’.
version	2	263	Always the ASCII string ‘00’ without terminating NULL character.
RFU	247	265	

3.4 Exporting Data by reading from TSE_TAR.xxx

The most efficient (and recommended) way to perform an unfiltered export of all transaction information is to simply read the TAR archive that is split across all *TSE_TAR.xxx* files and concatenate them to form the final export. On a POSIX system (e.g. a typical Linux system), the concatenation of the tar files can be achieved by the command

```
cat TSE_TAR.001 TSE_TAR.002 TSE_TAR.003 ... > TSE_TAR.tar
```

To reduce the number of bytes that have to be read, one can limit reading to *TAR Export Size* bytes (see Section 3.2), which is the size of the final TAR archive. Please note that this command only works if the file system cache on the host system is not active (see Section 5.6).

Note: For a filtered export, please refer to Section 4.6.

3.5 Sending commands to the TOE

Apart from standard file operations, there are special commands available to use the special features provided by the TOE. Commands are sent to the TOE by writing the command data to the beginning of the file *TSE_COMM.DAT*. The response is obtained by reading from the same location.

The handshake between host and TOE requires that every write to *TSE_COMM.DAT* needs to be followed by a mandatory read from the same location before the next write will have an effect. In case two consecutive writes are performed, the second write is discarded and also not acknowledged (by an incremented Write Index in the command response) unless the response has been read after the first write.

The basic command structure is shown in Table 3.3. A command starts with the length of the entire command and is followed by command specific data of at most 510 bytes. Together, this forms a command of exactly 512 bytes. After sending a command, the response can be fetched immediately. In case the response is less than 512 bytes long, the remaining bytes are filled with zeros to always create a response of exactly 512 bytes. The response structure is given in Table 3.4. The complete list of commands of the TOE can be found in Section 4.

Please note that verification whether a command is in execution or already executed requires checking the *Result Code* of the response with detection of the respective condition `0xFF`, `0xFD` or `0xFE`. Also, the *Write Index* must be checked before sending a command. This index must be incremented by exactly 1 in the command response to ensure that the response corresponds to the command that was just sent and not to a previous command (or no command at all after power cycling the device).

Table 3.3: Command Interface in *TSE_COMM.DAT*

Field	Size	Comment
Command Data Length	2	Length of Command Data.
Command Data	N	Command to execute. Maximum length is 510 Bytes.

Table 3.4: Command Response

Field	Size	Offset	Comment
Write Index	4	0	Write Index for acknowledgement, starts with 0 after each power cycle, incremented on a write to *TSE_COMM.DAT*. This index therefore serves to acknowledge the previous write and ensures the command response is exactly from this write. The handshake requires getting the transaction result after each write to *TSE_COMM.DAT*.
Result Code	1	4	**`0xFF`:** Command completed and response available. **`0xFE`:** Command not yet completed, continue to read *TSE_COMM.DAT* until value is `0xFF` or `0xFD`. **`0xFD`:** Command response available, please use command *Fetch Command Response* to fetch it.
Command Response Length	2	5	Length of command response.
Command Response	Up to 505	7	Response Payload incl. status word (last two bytes). Valid with Result Code `0xFF` only.

Warning: In this context it is essential to highlight the difference between the Result Code and the actual Command Response from the previous table. While the Result Code is a tool that only serves to control the flow of commands and to provide feedback, whether a command actually finished yet, the Command Response contains the actual feedback from a command.

The following image shows an example of a simple communication between the TOE and the client in order to visualize the principles explained before.

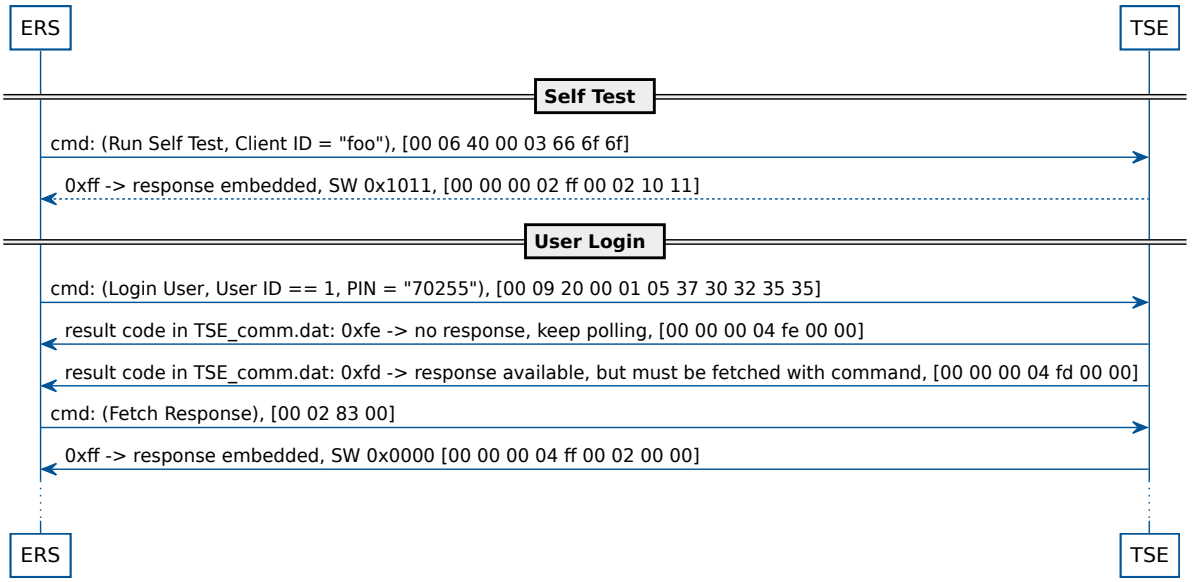

Fig. 3.3: Example for a communication with the TOE

Command Status Word

Each successful command execution leads to a Command Response, which contains at least 2 bytes of data. These two bytes are located at the end of the Command Response and together are called the command's status word (SW).

A SW of `0x0000` means the command executed without errors. Any other SW indicates an error condition. All possible SW are described in Table 3.5. The SW is part of every command's response and the possible values (other than `0x0000`) are described for each command in the following sections.

Table 3.5: Status Words of commands

`0x0000`: Execution successful
`0x1001`: Unspecified, internal processing error
`0x1002`: Time not set
`0x1004`: No transaction in progress
`0x1005`: Invalid command syntax
`0x1006`: Not enough data written during transaction
`0x1007`: Invalid parameter
`0x1008`: Given transaction is not started
`0x1009`: Maximum parallel transactions reached
`0x100A`: Certificate expired
`0x100C`: No last transaction to fetch
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x100E`: Signatures exceeded
`0x100F`: Not authorized
`0x1010`: Maximum registered clients reached
`0x1011`: Client not registered
`0x1012`: Failed to delete, data not completely exported
`0x1013`: Client has unfinished transactions
`0x1014`: TSE contains unfinished transactions
`0x1015`: Wrong state, no command response to fetch
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0x1050`: Wrong state, changed PUK required
`0x1051`: Wrong state, changed PIN required
`0x1053`: Wrong state, active CTSS interface required
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x1061`: Firmware Update: Integrity check failed
`0x1062`: Firmware Update: Decryption failed
`0x1064`: Firmware Update: Wrong format
`0x1065`: Firmware Update: Internal error
`0x1067`: Firmware Update: downgrade prohibited
`0x10FD`: Wrong state, TSE already initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x10FF`: Wrong state, TSE not initialized
`0x11xx`: Authentication failed, *xx* give the number of remaining retries
`0x1201`: PIN is blocked
`0x1202`: Given user is not authenticated
`0x1300`: Self test of FW failed
`0x1310`: Self test of CSP failed
`0x1320`: Self test of RNG failed
`0x1400`: Firmware Update: Base FW update error
`0x1500`: Firmware Update: FW Extension update error
`0x1600`: Firmware Update: CSP update error
`0x2001`: Filtered Export: no export in progress

Continued on next page

Table 3.5 – continued from previous page

`0x2002`: Filtered Export: no new data, keep polling
`0x2003`: Filtered Export: no matching entries, export would be empty
`0xF000`: Command not found
`0xFF00`: Signature creation error

CHAPTER

FOUR

COMMANDS

4.1 User Authentication Commands

The TOE manages different users, each having a PIN and a role.

PINs have a retry counter of 3. If a wrong PIN has been entered 3 times, the PIN will be blocked and must be unblocked with the PUK, which belongs to the *Admin* user.

The PUK also has a retry counter of 3. If the wrong PUK has been entered 3 times, the PUK will be blocked. This situation can not be recovered from. If the PINs are also blocked, the TOE can not be configured anymore and must be replaced.

Please note that the access control model of the TOE bases on a role model rather than individual users. While this command is actually used by users of the TOE (meaning entities external to the TOE), its primary purpose in the sense of access control is to provide the ability to add a certain role to the current user context. The TOE maintains a strict and static 1:1 relationship between the existing users and roles. More precisely, the role *Admin* only has one user as a member, namely the user *Admin* and the role *TimeAdmin* only has one member, the user *TimeAdmin*. While user *Admin* and role *Admin* are strictly speaking not identical, they can often be used interchangeably.

Some commands require a user with a specific role to be logged in in order to succeed. The following table summarizes the existing roles and their associated permissions.

Table 4.1: User Permissions

Id	Role	Permissions
0	Unidentified User	• *Login User* • *Logout User* • *Unblock User* • *Change PUK* • *Change PIN* (but the user has to be logged in) • *Run Self Test* • *Fetch Command Response* • *Get Last Transaction Response* • *List Started Transactions* • *Get Log Message Certificate* • *TSE Flash Information* • *Data Import Initialize* • *Data Import Finalize* • *Data Import Rollback* • *Start Filtered Export* • *Poll Filtered Export* • *Abort Filtered Export* • *Acknowledge Export*
1	Admin	Full access to all commands.
2	Time Admin	*Unidentified User* + *Update Time*

Beside the roles that are used to control the access of users to commands, the TOE also maintains a set of states as listed in the following table. These states indicate characteristics of the current overall state of the TOE and are used in some commands for access control purposes.

Table 4.2: States

State	Description
selfTestRun	This state is active if the self test has been executed at least once after booting the TOE (see Section 4.2.1).
selfTestPassed	This state is active if the self test of the TOE has been passed successfully. It is inactive if the self test failed. If the state is inactive, the TOE is in the *secure state* as defined in FPT_FLS.1 in [ST]. This state requires the state *selfTestRun* to be active.
CTSSInterfaceState	This state is active if a) the self test of the TOE has been passed AND b) the CTSS interface of the TOE has been enabled before with *Enable CTSS Interface*. Please note that this state is immediately made inactive if either the self test fails or the *Admin* disables the CTSS interface with *Disable CTSS Interface*.
TSEInitialized	This state is active if a) the TOE has been initialized with *Initialize TSE* AND b) the TOE has not been decommissioned with *Decommission TSE*, yet.

Note: The self test that is mentioned in the description of the CTSSInterfaceState includes the tests as described in FPT_TST.1 as well as FPT_TEE.1.

Note: Enabling and Disabling the CTSS interface via *Enable CTSS Interface* and *Disable CTSS Interface* persists over power cycles.

Note: [PP-SMAERS] defines two dedicated roles named CSP role and CTSS interface role. These roles have no direct equivalent in form of a role in the TOE as they do not require user authentication. Instead, the CTSSInterfaceState is the equivalent of the CTSS interface role and the selfTestPassed state is the equivalent of the CSP role.

Whenever *Login User*, *Logout User*, or *Unblock User* is invoked, a *Log Message* will be generated and stored in the *TSE Store* to log the result of the operation. For other *User Authentication Commands*, no log message will be generated.

4.1.1 Login User

Description

Authenticates users of the TOE based on their PIN.

After successful execution of this command by a user, the corresponding role will be added to the set of active roles, which might allow to execute privileged commands. For example: if the command is called with the *Admin's* User ID and is successfully executed, the user will be allowed to execute commands that require the *Admin* role afterwards. These changes to the role context are applied immediately after the command returns. If the command fails, the roles that are associated with the current user context do not change. In this case, the command returns [`0x11xx`: Authentication failed, *xx* give the number of remaining retries].

The current user context can be associated with multiple roles at the same time. If multiple users with different roles are logged in, the effective privileges are the union of all logged in roles (e.g. if *Admin* and *TimeAdmin* are logged in, the time can be set and administrative commands can be sent).

After a reboot of the TOE, all users are logged out again.

In case the provided PIN is wrong, the command will respond with [`0x11xx`: Authentication failed, *xx* give the number of remaining retries] and the retry counter is decreased. If the retry counter is currently 1 and the wrong PIN is used (thus the retry counter reaches 0), the number of remaining retries will be set to 0 and the SW will be `0x1100`. Afterwards, both an authentication with and without a valid PIN will return [`0x1201`: PIN is blocked]. In order to login the user again, the user must be unblocked with command *Unblock User*.

This command causes a Log Message to be signed and thus can only be executed if the CSP is still operational.

Note: The initial PINs are device dependent and can be calculated as described in *Initial PUK and PINs*.

Note: Before any user can be logged in, the initial PUK must have been changed (see Section 4.1.4), otherwise the command will fail with [`0x1050`: Wrong state, changed PUK required].

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.3: Command Data: Login User

Field	Size	Offset	Value	Comment
Command	2	0	20 00	
User Id	1	2	Id as given in Table 4.1 (except 0)	User to log in as.
PIN Length	1	3		Must be 5.
PIN	5	4		Byte array.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.4: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1007`: Invalid parameter
`0x1050`: Wrong state, changed PUK required
`0x1201`: PIN is blocked
`0x11xx`: Authentication failed, *xx* give the number of remaining retries
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.1.2 Logout User

Description

Logs out the given user. The user must be logged in, otherwise the command will fail with [`0x1202`: Given user is not authenticated].

On successful execution, the user's role will be immediately removed from the active roles and privileged commands might not be executable anymore. Please refer to *Login User* for more details about the relationship between users and their roles.

This command causes a Log Message to be signed and thus can only be executed if the CSP is still operational.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.5: Command Data: Logout User

Field	Size	Offset	Value	Comment
Command	2	0	21 00	
User Id	1	2	Id as given in Table 4.1	User to log out.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.6: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1202`: Given user is not authenticated
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.1.3 Unblock User

Description

This command serves two distinct purposes: On the one hand, the command can be used to unblock a user that has been blocked due to too many unsuccessful authentication attempts. On the other hand, the command can be used to change the PIN of a user. Therefore, it can be used to recover the credentials of a user in case of a forgotten PIN.

As both commands are administrative commands, the PUK must be provided.

The new PIN must be different from the previous one (even when just unblocking the user), otherwise the command will fail with [`0x1007`: Invalid parameter].

The PUK has an associated retry counter. In case the provided PUK is wrong, the response SW is [`0x11xx`: Authentication failed, *xx* give the number of remaining retries] and the retry counter is decreased. If the retry counter is currently 1 and the wrong PUK is used (thus the retry counter reaches 0), the number of remaining retries will be set to 0 and the SW will be `0x1100`. Afterwards, both an authentication with and without a valid PUK will return [`0x1201`: PUK is blocked]. As a blocked PUK can not be recovered from, it is recommended to export all data and decommission the TOE. Afterwards, a new TSE should be used.

This command causes a Log Message to be signed and thus can only be executed if the CSP is still operational.

Note: Before this command can be executed, the initial PIN of the user must have been changed (see Section 4.1.5), otherwise the command will fail with [`0x1051`: Wrong state, changed PIN required].

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.7: Command Data: Unblock User

Field	Size	Offset	Value	Comment
Command	2	0	22 00	
User Id	1	2	Id as given in Table 4.1 (except 0)	User to unblock.
PUK Length	1	3		Must be 6.
PUK	6	4		Byte array.
New PIN Length	1	10		Must be 5.
New PIN	5	11		Byte array.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.8: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1007`: Invalid parameter
`0x1051`: Wrong state, changed PIN required
`0x1201`: PUK is blocked
`0x11xx`: Authentication failed, *xx* give the number of remaining retries
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.1.4 Change PUK

Description

This command can be used to change the *Admin* PUK.

The new PUK must be different from the previous PUK, otherwise the command fails with [`0x1007`: Invalid parameter].

The PUK has an associated retry counter. In case the provided PUK is wrong, the response SW is [`0x11xx`: Authentication failed, *xx* give the number of remaining retries] and the retry counter is decreased. If the retry counter is currently 1 and the wrong PUK is used (thus the retry counter reaches 0), the number of remaining retries will be set to 0 and the SW will be `0x1100`. Afterwards, both an authentication with and without a valid PUK will return [`0x1201`: PUK is blocked]. As a blocked PUK can not be recovered from, it is recommended to export all data and decommission the TOE. Afterwards, a new TSE should be used.

Note: The PUK can be changed even if the user *Admin* is not logged in, which means that the *Admin* PIN is not required to change the PUK.

Note: The initial PUK is device dependent and can be calculated as described in *Initial PUK and PINs*.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.9: Command Data: Change PUK

Field	Size	Offset	Value	Comment
Command	2	0	23 00	
Current PUK Length	1	2		Must be 6.
Current PUK	6	3		Byte array.
New PUK Length	1	9		Must be 6.
New PUK	6	10		Byte array.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.10: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1007`: Invalid parameter
`0x1201`: PUK is blocked
`0x11xx`: Authentication failed, *xx* give the number of remaining retries

4.1.5 Change PIN

Description

Changes the PIN of the given user. The user must have been logged in before with command *Login User*, otherwise the command will fail with [`0x1202`: Given user is not authenticated].

In order to change the PIN, the current PIN must be provided as well as the new PIN, which must be different from the current PIN (otherwise the command fails with [`0x1007`: Invalid parameter]).

The PIN has an associated retry counter. In case the provided PIN is wrong, the response SW is [`0x11xx`: Authentication failed, *xx* give the number of remaining retries] and the retry counter is decreased. If the retry counter is currently 1 and the wrong PIN is used (thus the retry counter reaches 0), the number of remaining retries will be set to 0 and the SW will be `0x1100`. Afterwards, a PIN change with and without a valid PIN will return [`0x1201`: PIN is blocked].

If users were blocked by this command, they can be unblocked with command *Unblock User*.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.11: Command Data: Change PIN

Field	Size	Offset	Value	Comment
Command	2	0	24 00	
User Id	1	2	Id as given in Table 4.1	User to change the PIN for.
Current PIN Length	1	3		Must be 5.
Current PIN	5	4		Byte array.
New PIN Length	1	9		Must be 5.
New PIN	5	10		Byte array.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.12: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1007`: Invalid parameter
`0x1202`: Given user is not authenticated
`0x1201`: PIN is blocked
`0x11xx`: Authentication failed, *xx* give the number of remaining retries

4.2 Self Test Commands

4.2.1 Run Self Test

Description

After each power cycle, the TOE runs a self test to ensure proper operation of its internal modules. The self test consists of three parts:

1. Test of the TOE itself (e.g. data consistency). This part of the self test includes a health test of the random number generator. If this test fails, the command fails with [`0x1300`: Self test of FW failed] or with [`0x1320`: Self test of RNG failed] in the case that the source of the error is the RNG.
2. Test of the CSP. If this test fails, the command fails with [`0x1310`: Self test of CSP failed].
3. Test of the ERS. If this test fails, the command fails with [`0x1011`: Client not registered].

Since the self test depends on the *Client ID* provided by the ERS, the self test can only be completed successfully by issuing this command and thus must be run as first command after the TOE boots, otherwise no other command can be executed. The client must have been registered before with command *Register Client*, otherwise this command will fail with [`0x1011`: Client not registered].

The self test can be repeated whenever it is desired by the ERS, but it must be run at least once every 25 hours. Otherwise, the TOE will set the state *selfTestRun* to inactive, which makes all future commands fail until the self test is run successfully again. The time until the *selfTestRun* state will be made inactive can be retrieved as *Time Until Next Selftest* from *TSE Status*.

The self test is a potentially long running operation that might take up to 60 seconds to complete.

Please note that the CSP gets power cycled during the self test and thus the internal time will be set back to zero and must be set again after a successful self test.

Permissions

None.

Command structure and parameters

Table 4.13: Command Data: Run Self Test

Field	Size	Offset	Value	Comment
Command	2	0	40 00	
Client ID Length	1	2	L1	Maximum length is 30 bytes.
Client ID	L1	3		ASCII string representing the unique serial number of the client.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.14: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1007`: Invalid parameter
`0x1011`: Client not registered
`0x1300`: Self test of FW failed
`0x1310`: Self test of CSP failed
`0x1320`: Self test of RNG failed

4.2.2 Register Client

Description

Registers a client (i.e. an ERS) as a valid system for self tests and transactions. A client is identified by its ID, which shall be a unique string (e.g. its serial number). If the same client is already registered, the command will be successful, but the client will not be registered twice.

The amount of currently and maximally registered clients can be obtained from *TSE Status*. If this number has been reached and the command is executed again, it will fail with [`0x1010: Maximum registered clients reached`].

Note: Before this command can be executed, the initial PIN for each user must have been changed (see Section 4.1.5), otherwise the command will fail with [`0x1051: Wrong state, changed PIN required`].

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.15: Command Data: Register Client

Field	Size	Offset	Value	Comment
Command	2	0	41 00	
Client ID Length	1	2	L1	Maximum length is 30 bytes.
Client ID	L1	3		ASCII string representing the unique serial number of the client. The string must be representable as an ASN.1 PrintableString, which restricts the allowed characters to the following: `A-Za-z0-9'()+,-./:=?` and the space character.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000: Execution successful`], otherwise one of the following error codes is returned.

Error Codes

Table 4.16: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized
`0x1007`: Invalid parameter
`0x1051`: Wrong state, changed PIN required
`0x1010`: Maximum registered clients reached

4.2.3 Deregister Client

Description

Removes a client from the list of authorized clients.

In case the client is not registered, this command will fail with [`0x1011`: Client not registered].

Before a client can be deregistered, all transactions belonging to that client must be finished first, otherwise the command will fail with [`0x1013`: Client has unfinished transactions]. An unfinished transaction always belongs to the client that updated it most recently (or started the transaction in case it was never updated at all).

Please be aware that for passing the self test (see Section 4.2.1) at least one client must be registered.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.17: Command Data: Deregister Client

Field	Size	Offset	Value	Comment
Command	2	0	42 00	
Client ID Length	1	2	L1	Maximum length is 30 bytes.
Client ID	L1	3		ASCII string representing the unique serial number of the client.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.18: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized
`0x1007`: Invalid parameter
`0x1011`: Client not registered
`0x1013`: Client has unfinished transactions

4.2.4 List Registered Clients

Description

Lists all registered clients in chunks of 16 clients.

By providing a non-zero value for *Client Offset*, this amount of clients can be skipped from the beginning of the returned list. For example, a value of 0 will return the first 16 registered clients and a value of 3 will return the 4th to 19th registered clients.

The *Amount* field in the response gives the amount of client IDs that are stored in the response. If this is smaller than 16, then there are no further clients registered.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.19: Command Data: List Registered Clients

Field	Size	Offset	Value	Comment
Command	2	0	43 00	
Client Offset	4	2		The amount of registered clients to skip.

Response

In case of success, the SW is [`0x0000: Execution successful`] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.20: Response Data: List Registered Clients

Field	Size	Offset	Value	Comment
Amount	1	0	n	Amount of clients in this response. 0 <= n <= 16.
Client ID 1	31	1		NULL terminated ASCII string of 1st registered client ID.
Client ID 2	31	32		NULL terminated ASCII string of 2nd registered client ID.
...	...	...		
Client ID 16	31	466		NULL terminated ASCII string of 16th registered client ID.

Error Codes

Table 4.21: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized

4.3 Maintenance Commands

4.3.1 Enable CTSS Interface

Description

This command can be used to enable the CTSS Interface. An enabled CTSS interface is a pre-requisite for many other commands that are described in this specification.

The current status of the CTSS interface can be obtained by reading bit 2 from the *TSE Security* value from *TSE Status*.

Note: The CTSS interface refers to the logical interface that operates on top of the file based interface as described in *File Based Interface*. It can also be referred to as ERS interface.

By default, the CTSS interface is disabled, which means that no transactions can be performed and no data can be read from the *TSE Store*.

The setting that is changed by this command is persisted across power cycles.

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.22: Command Data: Enable CTSS Interface

Field	Size	Offset	Value	Comment
Command	2	0	60 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.23: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized

4.3.2 Disable CTSS Interface

Description

This command can be used to disable the CTSS Interface. An enabled CTSS interface is a pre-requisite for many other commands that are described in this specification.

The current status of the CTSS interface can be obtained by reading bit 2 from the *TSE Security* value from *TSE Status*.

After disabling the interface, commands that require the *CTSSInterfaceState* to be active can not be executed anymore. Thus, when sending the TSE to maintenance, it is recommended to disable the CTSS interface in order to prevent reading the recorded transactions.

The setting that is changed by this command is persisted across power cycles.

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.24: Command Data: Disable CTSS Interface

Field	Size	Offset	Value	Comment
Command	2	0	61 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.25: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized

4.3.3 Initialize TSE

Description

Before the TOE can be used for transactions, it must be initialized with this command.

By initializing the TOE, the user takes ownership of the TOE and activates the *TSEInitialized* state (see Section 4.1), which allows commands that need this state to be active to be executed.

Note: This command can only be issued once during the life time of the TOE. If the command is called again afterwards, it will fail with [`0x10FD`: Wrong state, TSE already initialized].

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.26: Command Data: Initialize TSE

Field	Size	Offset	Value	Comment
Command	2	0	70 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.27: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FD`: Wrong state, TSE already initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x100F`: Not authorized
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.3.4 Decommission TSE

Description

When the TOE should not be used anymore, it must be decommissioned with this command.

Successful execution of this command will permanently remove the ability to store new transactions in the TOE as the CSP can no longer perform signatures afterwards. After issuing this command, the *TSEInitialized* state (see Section 4.1) will be deactivated permanently and command *Initialize TSE* will be blocked to prevent re-initialization of the TSE.

Decommissioning is only allowed if there are no unfinished transactions, otherwise the command will fail with [`0x1014`: TSE contains unfinished transactions].

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).
- The time must have been set before (see Section 4.3.5).

Command structure and parameters

Table 4.28: Command Data: Decommission TSE

Field	Size	Offset	Value	Comment
Command	2	0	71 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.29: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x1002`: Time not set
`0x100F`: Not authorized
`0x1014`: TSE contains unfinished transactions
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.3.5 Update Time

Description

After each power cycle, the *TSE Store* is locked and no transactions are possible until the time of the ERS has been synchronized with the time of the TOE using this command.

The TOE will forward the timestamp to its CSP and will use this time to properly timestamp Log Messages. Depending on the accuracy of the CSP's internal clock, this command must also be called regularly to keep the host and TOE time synchronized. How often the time must be synchronized is announced in *TSE Status*. Applications should take care to not synchronize the time too frequently as this negatively effects the endurance of the CSP. It is thus recommended to synchronize the time as close as possible to the interval that is announced in *TSE Status*.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).
- This command requires the user *Admin* or *Time Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.30: Command Data: Update Time

Field	Size	Offset	Value	Comment
Command	2	0	80 00	
Timestamp	8	2		Timestamp as seconds since Unix Epoch. The timestamp will be interpreted as an unsigned number, which means only dates after 1970 are supported. Big Endian.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.31: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x100F`: Not authorized
`0x100A`: Certificate expired
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*
`0xFF00`: Signature creation error

4.3.6 TSE Firmware Update Transfer

Description

Transfers a firmware update package to the TOE.

Since the firmware package can be quite large, it must be transferred in multiple chunks. The first chunk will be transmitted with a *Chunk Offset* set to 0 and an arbitrary *Chunk Length* L1. The next chunk will be transmitted with a *Chunk Offset* equal to L1 and a *Chunk Length* of L2. Another chunk will be transferred with a *Chunk Offset* of L1 + L2 and a *Chunk Length* of L3 and so on until the final chunk has been transferred. To then apply the update, call command *TSE Firmware Update Apply*.

If the *Chunk Offset* is bigger than the reserved space for a firmware update package, this command will fail with [`0x1007`: Invalid parameter]. Additionally, *Chunk Offset* and *Chunk Length* must be multiples of 16, otherwise this command will fail with [`0x1007`: Invalid parameter].

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.32: Command Data: TSE Firmware Update Transfer

Field	Size	Offset	Value	Comment
Command	2	0	50 00	
Chunk Offset	4	2		Offset in the firmware package where *Chunk Data* is stored. Big Endian.
Chunk Length	2	6	L1	Size of the current chunk in bytes. Big Endian.
Chunk Data	L1	8		Raw data of the current chunk.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.33: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized
`0x1007`: Invalid parameter

4.3.7 TSE Firmware Update Apply

Description

Applies a firmware update that was previously transferred to the TOE with command *TSE Firmware Update Transfer*. The firmware will be checked by the CSP for authenticity and integrity before being applied.

Please note that this is a long running operation, which – depending on the size of the firmware update – might take several minutes to complete. It must be ensured that there is no power loss while applying the firmware update, as this might brick the device and make it unusable. Therefore it is recommended to export all data before applying a firmware update. However, only the last seconds of the firmware update process are critical and must not be interrupted, losing power anywhere prior does not affect the TOE at all.

In case the firmware update succeeds, the TSE automatically performs a power cycle. In that case the *Write Index* in the response will be set to `0`. Since this situation can not be distinguished from a power cycle that happened randomly during the update process, the ERS should read the currently installed *TSE Software Version* (see Section 3.2) before applying the update and read it again after the update has been completed. If the new value is numerical bigger than the old value, the update was successfully applied.

If the *Firmware Update Package Size* is 0 or not a multiple of 512, this command will fail with [`0x1007`: Invalid parameter]. If the decrypted firmware package can not be parsed, this command will fail with [`0x1064`: Firmware Update: Wrong format]. If the firmware package to be installed does not have a higher version number as the one currently installed, this command will fail with [`0x1067`: Firmware Update: downgrade prohibited] to prevent downgrading the TOE to an earlier version.

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.34: Command Data: TSE Firmware Update Apply

Field	Size	Offset	Value	Comment
Command	2	0	51 00	
Firmware Update Package Size	4	2		Total size of the firmware package that has been transferred with *TSE Firmware Update Transfer*. Big Endian.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.35: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized
`0x1007`: Invalid parameter
`0x1061`: Firmware Update: Integrity check failed
`0x1062`: Firmware Update: Decryption failed
`0x1064`: Firmware Update: Wrong format
`0x1065`: Firmware Update: Internal error
`0x1067`: Firmware Update: downgrade prohibited
`0x1400`: Firmware Update: Base FW update error
`0x1500`: Firmware Update: FW Extension update error
`0x1600`: Firmware Update: CSP update error

4.3.8 Enable Export If CSP Test Fails

Description

The TOE allows to determine the behavior of the TOE with respect to the export of data if the CSP test fails during the self test.

By default, no data can be exported anymore if the CSP test fails due to a broken security module. To allow data to be exportable if the CSP test fails, this command can be used. Please note that this command will only allow to do a complete export (see Section 3.3) of the tar archive; the export commands given in Section 4.6 will still be disabled. Also, all commands of the TOE that require a successful self test for their execution will still be inaccessible.

This command can only be used while the CSP self test is still passing. As soon as the test fails, it is too late to change this setting and this command will fail.

This setting is persisted across power cycles.

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.36: Command Data: Enable Export If CSP Test Fails

Field	Size	Offset	Value	Comment
Command	2	0	62 01	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.37: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized

4.3.9 Disable Export If CSP Test Fails

Description

The TOE allows to determine the behavior of the TOE with respect to the export of data if the CSP test fails during the self test.

To disable the functionality of export and prevent any data from being exported if the CSP test fails, use this command. Please note that in case of a broken CSP, the data on the TOE is then lost and can not be recovered. This is the factory default behavior.

This setting can only be changed while the CSP test is still passing. As soon as the test fails, it is too late to change this setting and this command will fail.

This setting is persisted across power cycles.

Permissions

- This command requires the state *selfTestPassed* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).

Command structure and parameters

Table 4.38: Command Data: Disable Export If CSP Test Fails

Field	Size	Offset	Value	Comment
Command	2	0	62 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.39: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x100F`: Not authorized

4.4 Utility Commands

4.4.1 Fetch Command Response

Description

This command must be used if and only if the *Result Code* of a previously executed command is `0xFD`. It will then deliver the response of the previously issued command.

No further commands will be accepted by the TOE if the last command's *Result Code* was *0xFD* until *Fetch Command Response* is issued. In that case, any other command will simply be ignored, which can be detected as the *Write Index* in the command response is not increased then.

Note: If the command's *Result Code* is `0xFF`, then the actual response is already embedded in the currently read response and does not need to be explicitly fetched. This command is only needed if the *Result Code* is `0xFD`.

Permissions

This command does not require any permissions, as it is only a helper to deliver the actual response data of the last issued command (which was already checked for necessary permissions during its execution).

Command structure and parameters

Table 4.40: Command Data: Fetch Command Response

Field	Size	Offset	Value	Comment
Command	2	0	83 00	Delivers Command Response of previous command.
Data	<empty>			

Response

The response can be any of the other command's responses.

Error Codes

If there is a response to fetch, this command contains the error code that was set by the previously executed command.

If there is no response to fetch, i.e. this command gets executed when the last command's *Result Code* is NOT `0xFD`, one of the following errors will be returned.

Table 4.41: Error Codes

`0x1015`: Wrong state, no command response to fetch

4.4.2 Get Last Transaction Response

Description

This command can be used to query the last transaction's response.

Optionally, instead of returning the newest transaction response, the last transaction response that was created by a specific client can be queried by providing a non zero-length *Client ID* to filter for.

This command is useful in case the ERS loses track of the last transaction result (e.g. because it crashes or loses power). In that case, the ERS might not know if the last executed transaction was properly finalized or not and can query the last transaction's response with this command to sync its internal state with the TOE state.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.42: Command Data: Get Last Transaction Response

Field	Size	Offset	Value	Comment
Command	2	0	84 00	
Client ID Length	1	2	L1	Maximum length is 30 bytes.
Client ID	L1	3		ASCII string representing the unique serial number of the client. Use a zero length string to ignore this parameter and return the newest transaction response, no matter which client created it.

Response

In case of success, the SW is [`0x0000`: Execution successful] and the response data fields are the same as for command *Data Import Finalize*. Otherwise, one of the following error codes is returned.

Error Codes

Table 4.43: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x100C`: No last transaction to fetch

4.4.3 List Started Transactions

Description

Lists all started transaction numbers in chunks of 62 transactions.

By providing a non-zero value for *Transaction Offset*, this amount of transaction numbers can be skipped from the beginning of the returned list. For example, a value of 0 will return the first 62 started transaction numbers and a value of 3 will return the 4th to 65th transaction numbers.

The *Amount* field in the response gives the amount of transaction numbers that are stored in the response. If this is smaller than 62, then there are no further started transactions.

Optionally, only transactions belonging to a specific client can be queried by providing a non zero-length *Client ID* to filter for. An unfinished transaction always belongs to the client that updated it most recently (or started the transaction in case it was never updated at all). That means that if a transaction is started by client A and then updated by client B, this command will return this specific transaction number only when filtering for transactions belonging to client B (or when not filtering at all), not when filtering for transactions belonging to client A, because client B was the last client that updated the transaction.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.44: Command Data: List Started Transactions

Field	Size	Offset	Value	Comment
Command	2	0	85 00	
Transaction Offset	4	2		The amount of started transactions to skip.
Client ID Length	1	6	L1	Maximum length is 30 bytes.
Client ID	L1	7		ASCII string representing the unique serial number of the client. Only open transactions belonging to this client will be returned. Use a zero length string to return all open transactions without further information to which client the transactions belong.

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.45: Response Data: List Started Transactions

Field	Size	Offset	Value	Comment
Amount	1	0	n	Amount of transaction numbers in this response. 0 <= n <= 62. If this is < 62, then there are no further started transactions.
Transaction Number 1	8	1		Transaction number of 1st started transaction.
Transaction Number 2	8	9		Transaction number of 2nd started transaction.
...	...	...		
Transaction Number 62	8	489		Transaction number of 62nd started transaction.

Error Codes

Table 4.46: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned

4.4.4 Get Log Message Certificate

Description

Returns the certificate that is associated with the signatures created by the TOE. This certificate can be used to verify the signatures of all Log Messages created by the TOE.

The returned data is a single PEM file, which contains the complete certificate chain.

To verify a signature, only the leaf certificate (the first one in the PEM file) is required. However, in order to ensure that the certificate whose key has been used stems from the correct PKI, the certificate chain shall be verified back to the root of the PKI. Please refer to [AGD] for more details on how the root key of the PKI can be obtained.

Since the whole data might not fit into one response block, a *Data Offset* must be provided to select which parts of the certificate file should be returned. In case this number is equal to or bigger than the stored certificate file, the command will fail with [`0x1007`: Invalid parameter].

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.47: Command Data: Get Log Message Certificate

Field	Size	Offset	Value	Comment
Command	2	0	86 00	
Data Offset	4	2		Selects from which offset the certificate data should be returned. Big Endian.

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.48: Response Data: Get Log Message Certificate

Field	Size	Offset	Value	Comment
Certificate Data Length	2	0	R1	Length of the certificate data in this response block. If this is `0`, the end of the certificate data has been reached. Big Endian.
Certificate Data	R1	2		

Error Codes

Table 4.49: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x1007`: Invalid parameter

4.4.5 TSE Flash Information

Description

Provides low level information about the flash storage.

This command can be used to monitor the flash storage health and detect possible future defects before they occurr and apply predective maintenance.

As a recommendation, the following simple guidance is provided:

1. If *Uncorrectable ECC errors* is different from 0, the TSE should be replaced.
2. If *Percentage Remaining Spare Blocks All* gets below 25%, the TSE should be replaced.
3. If the average erase count (calculated as *Block Erases* / (*Flash Block Count* * 256)) is bigger than 2940, the TSE should be replaced.

Please note that based on the use case of the TSE, which does not involve many flash read or write operations compared to other use cases, it is not expected that any of these conditions will ever be fulfilled during the lifetime of the TSE.

The lowest wear level class (WL) and highest wear level class (WH) fields give the range of wear level classes that are currently in use. Blocks that are not subject to the wear leveling are not counted. The wear level threshold (T) gives the size of a wear level class, minus 1, in units of flash memory block erases. Thus, the number of block erases that the flash blocks have seen is between WL*(T+1) and WH*(T+1)-1.

A spare block is a flash block that will used as a replacement for defect blocks.

Permissions

- This command requires the state *selfTestRun* to be active (see Section 4.1).

Command structure and parameters

Table 4.50: Command Data: TSE Flash Information

Field	Size	Offset	Value	Comment
Command	2	0	12 00	

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Note: Please note that some of the information that is returned by this command contains proprietary information that needs detailed knowledge about the internal structure of the TSE for interpretation. If you should need more detailled information about these data fields, please contact your Swissbit representative.

Table 4.51: Response Data: TSE Flash Information

Field	Size	Offset	Value	Comment
Proprietary 1	25	0		Manufacturer proprietary format.
Defect Blocks	2	26		Number of manufacturer marked defect blocks. Big Endian.
Initial Spare Blocks Worst	2	28		Number of initial spare blocks (worst interleave unit). Big Endian.

Continued on next page

Table 4.51 – continued from previous page

Field	Size	Offset	Value	Comment
Initial Spare Blocks Sum	2	30		Number of initial spare blocks (sum over all interleave units). Big Endian.
Percentage Remaining Spare Blocks Worst	1	32		Percentage of remaining spare blocks (worst interleave unit).
Percentage Remaining Spare Blocks All	1	33		Percentage of remaining spare blocks (all interleave units).
Uncorrectable ECC errors	2	34		Number of uncorrectable ECC errors (not including startup ECC errors). Big Endian.
Correctable ECC errors	4	36		Number of correctable ECC errors (not including startup ECC errors). Big Endian.
Lowest wear level class	2	40		Big Endian.
Highest wear level class	2	42		Big Endian.
Wear level threshold	2	44		Big Endian.
Block Erases	6	46		Total number of block erases. Big Endian.
Flash Block Count	2	52		Number of flash blocks, in units of 256 blocks. Big Endian.
Erase Count Target	2	54		Maximum flash block erase count target, in wear level class units. Big Endian.
Power on count	4	56		Big Endian.
Proprietary 2	100	60		Manufacturer proprietary format.

Error Codes

Table 4.52: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first

4.5 Transaction Commands

[BSI-TR-03153] defines three types of operations for transactions – Start, Update, Finish. Since the Process Data of a transaction might be bigger than the payload a single command can handle, each transaction type is *additionally* split into three phases: Initialize, Transfer, Finalize. Together, these phases form a single `Start-`, `Update-`, or `FinishTransaction`.

Fig. 4.1 shows how these phases are combined to form a single *Start Transaction* operation.

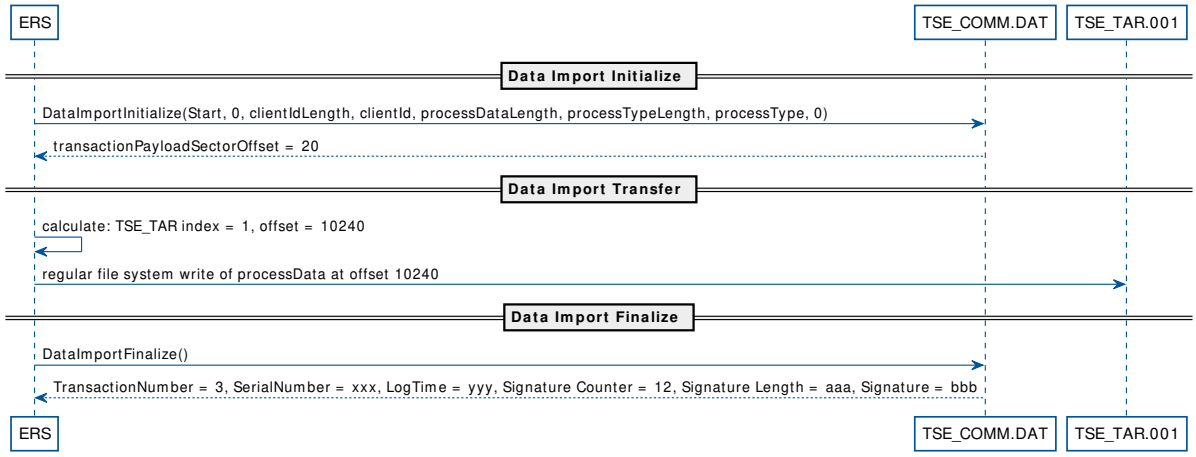

Fig. 4.1: Sequence to perform a complete *Start Transaction*

During *Initialize*, the transaction is initiated by sending the command *Data Import Initialize*, which carries all the metadata of the transaction (including whether the transaction is a *Start*, *Update*, or *Finish*). The response contains a relative sector offset which must be used to write the *Process Data* of the transaction directly into the *TSE Store* in the *Transfer* phase.

During *Transfer*, the Process Data of the transaction gets written consecutively to the *TSE Store* (which is made writable for these sectors) starting at the sector offset given in the response of *Data Import Initialize*. For performance reasons, this phase does not use commands, but writes the data to the *TSE Store* directly using regular file write commands (see Section 4.5.2).

After all data has been sent, the transaction can be made persistent with the command *Data Import Finalize*, which forms the *Finalize* phase of a transaction.

Note: [BSI-TR-03116-5] requires the ERS to update a started transaction (transaction with type *Transaction Start*) at most *MAX_UPDATE_DELAY* seconds after the new Process Data has been created on the cash register. The value of *MAX_UPDATE_DELAY* is announced in *TSE Status*. It is the responsibility of the cash register to comply with this requirement, it is not enforced by the TOE in any way.

Note: A transaction is only persisted after *Data Import Finalize* was executed successfully. If the TSE loses power after initializing a transaction but before finalizing it, it will be silently discarded on the next boot and must be resubmitted.

Note: As long as the transaction has not yet been finalized with command *Data Import Finalize*, it can be discarded with command *Data Import Rollback*.

High-Speed Mode

This mode behaves as described above. This is the preferred mode of operation, since it provides the easiest interface and the highest speed.

Simple Mode

Some ERS have technical limitations that restrict them from writing data at high offsets. Specifically for these ERS, the simple mode has been developed. This mode is activated per transaction by setting the corresponding flag in *Data Import Initialize*.

In this mode, instead of the relative sector offset where the Process Data is to be written, the response of *Data Import Initialize* always returns sector offset 0. All Process Data can then be written directly to the *TSE Store* at offset 0 in a consecutive stream. Internally, the TOE copies the provided data to the correct position. While the Data Import is in progress (i.e. it has not been finalized, yet), the Process Data can be read back from the same addresses. After *Data Import Finalize* has been successfully executed, the transaction is persisted and its data will only be readable from the real position in the *TSE Store* and will not be available anymore at offset 0.

Warning: This is **not** the preferred mode of operation and is much slower than the High-Speed Mode. This interface is made available for POS terminals that cannot seek to huge offsets inside a file.

Warning: While doing a transaction using the Simple Mode, reading from the *TSE Store* at addresses that are not part of the currently executed transaction is not allowed and will return blocks filled with zeroes. Use High-Speed Mode if you need to do transactions and read data at the same time.

4.5.1 Data Import Initialize

Description

Initializes a Data Import. By using command `90 00`, the Data Import will be performed in High-Speed mode (preferred), with command `91 00` Simple Mode will be used.

The parameter *Transaction Type* selects whether this Data Import shall start, update, or finish a transaction. For updating or finishing a transaction, the *Transaction Number* of a started transaction must be provided, otherwise this value must be set to 0. If the provided *Transaction Number* is not in the started state, the command will fail with [`0x1008`: Given transaction is not started].

The provided *Client ID* must have been previously registered (see Section 4.2.2), otherwise the operation will be rejected.

All registered clients can update or finish transactions, even transactions that have been started by another client. In case a transaction gets updated by another client as the last update (or the start of the transaction if the transaction has never been updated before), ownership of the transaction gets transferred to the new client.

The actual *Process Data* is omitted from the command and will be supplied afterwards (see Section 4.5.2). However, its length must be provided as *Process Data Length*.

The allowed values for *Process Type* will be defined by Kassensicherungsverordnung. They are not evaluated by the TOE and will be transparently copied into the generated Log Message.

The response field *Transaction Payload Offset* gives the sector offset in the *TSE Store* where the Process Data must be written during phase *Data Import Transfer* afterwards.

Note: After issuing this command, all commands except of *Fetch Command Response*, *Data Import Finalize*, and *Data Import Rollback* are not allowed and will fail with [`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed].

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).
- The time must have been set before (see Section 4.3.5).

Command structure and parameters

Table 4.53: Command Data: Data Import Initialize

Field	Size	Offset	Value	Comment
Command	2	0	9x 00	x = 0: High-Speed Mode Transactions x = 1: Simple Mode Transactions
Transaction Type	1	2	0: Transaction Start 1: Transaction Update 2: Transaction Finish	
Client ID Length	1	3	L1	Maximum length is 30 bytes.
Client ID	L1	4		ASCII string representing the unique serial number of the client.
Transaction Number	8	4 + L1	From previous response of *Transaction Start*.	Only valid for *Transaction Update* and *Transaction Finish*. Must be 0 for *Transaction Start*.

Continued on next page

Table 4.53 – continued from previous page

Field	Size	Offset	Value	Comment
Process Data Length	8	12 + L1		Big Endian.
Process Type Length	8	20 + L1	L2	Maximum length is 100 bytes.
Process Type	L2	28 + L1		ASCII string. The string must be representable as an ASN.1 PrintableString, which restricts the allowed characters to the following: `A-Za-z0-9'()+,-./:=?` and the space character.
Additional Data Length	8	28 + L1 + L2	L3	Must be 0 in the current version.
Additional Data	L3	36 + L1 + L2		

Note: As described in the previous table, the current TOE does not support the handover of additional data (which is an optional field in the definitions of [BSI-TR-03151]).

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.54: Response Data: Data Import Initialize

Field	Size	Offset	Value	Comment
Transaction Payload Sector Offset	8	0		Big Endian.

Error Codes

Table 4.55: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x1002`: Time not set
`0x100E`: Signatures exceeded
`0x1007`: Invalid parameter
`0x1011`: Client not registered
`0x1008`: Given transaction is not started
`0x1009`: Maximum parallel transactions reached
`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*

4.5.2 Data Import Transfer

Description

After successfully initializing the transaction, its payload (Process Data) must be written directly into the *TSE Store* at sector offset *Transaction Payload Sector Offset*. The corresponding range of sectors will be made writable by the TOE during transaction processing.

The *TSE Store* is a single continously allocated storage space. The *Transaction Payload Sector Offset* is the relative sector offset into this area. If the TSE is used in RAW access mode, the transaction data can thus simply be written with standard file write commands at the relative sector offset plus the offset of the *TSE Store* itself to the raw block device (see Section 5.1).

When using the TSE through the file system, the `TSE_TAR.xxx` files allow to write to the *TSE Store* (which is split into multiple smaller files because of file system limitations). To transfer the data to the TSE, it can be written to these files with regular file operations. For example, a sector offset of 20 means that the data must be written into file `TSE_TAR.001` at offset 10240 (20*512), a sector offset of 2097252 means the data must be written into file `TSE_TAR.002` at offset 51200 (100*512), because `TSE_TAR.001` can only fit 1 GB of data and offset 2097252 is bigger than 1 GB. The index of the file to use for transferring the data can be calculated with (*Transaction Payload Sector Offset* / 2097152) + 1. The offset into this file is calculated by *Transaction Payload Sector Offset* modulo 2097152. Care must be taken if the transaction would cross two `TSE_TAR` files. In that case, the transaction data must be split into two parts. The first one must be sent to the file `TSE_TAR.n` starting at the calculated offset, but only up to reaching the 1 GB file size limit, and the second one to `TSE_TAR.n+1` starting at offset 0. The transaction data will be part of the same transaction in the end as the *TSE Store* is a continous storage space internally.

While the Data Import is in progress (i.e. it has not been finalized, yet), the Process Data can be read back from the same addresses. After *Data Import Finalize* has been successfully executed, the transaction is persisted and its data will not be readable from the same addresses anymore, because it will be copied into the generated Log Message, which has a different layout.

If no Data Import has been initialized, all data that is written to the *TSE Store* will be discarded and will not be stored.

Note: Since this phase does not use commands, there are no response or return values available. Errors will be returned during *Data Import Finalize*.

4.5.3 Data Import Finalize

Description

After the TSE has received *Process Data Length* bytes as announced during *Data Import Initialize*, the transaction can be finalized, which will generate a signed Log Message of the transaction.

Please note that after calling this command, the data that has been sent in Section 4.5.2 will not be readable again from the same addresses, because it will be copied into a Log Message.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).
- The time must have been set before (see Section 4.3.5).

Command structure and parameters

Table 4.56: Command Data: Data Import Finalize

Field	Size	Offset	Value	Comment
Command	2	0	95 00	
Data	<empty>			

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.57: Response Data: Data Import Finalize

Field	Size	Offset	Value	Comment
Transaction Number	8	0		For *Transaction Start*: the newly assigned transaction number. For other types: the same transaction number that was used in the Initialize step.
Serial Number	32	8		Serial Number of the recording device. This is a hash over the public key / certificate of the Smart Card.
Log Time	8	40		Timestamp as seconds since Unix Epoch. The timestamp will be interpreted as an unsigned number, which means only dates after 1970 are supported. Big Endian.
Signature Counter	8	48		
Signature Length	8	56	R1	
Signature	R1	64		

Error Codes

Table 4.58: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x1002`: Time not set
`0x1004`: No transaction in progress
`0x1006`: Not enough data written during transaction
`0x100A`: Certificate expired
`0xFF00`: Signature creation error

4.5.4 Data Import Rollback

Description

In case there are any errors on the host while performing a Data Import (e.g. the host application crashes), host and TOE might go out of sync and the import cannot be completed successfully.

In that case, the import can be rolled back, which clears all pending data from the *TSE Store* and allows a new import to be started. The TOE will behave as if *Data Import Initialize* was never called.

Rolling back a Data Import is only possible before the Log Messages has been generated and signed during *Data Import Finalize*. Afterwards, the Data Import was already persisted and is thus not allowed to be rolled back.

If there is no Data Import in progress while calling this command, it will still return with [`0x0000`: Execution successful], but have no effect.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.59: Command Data: Transaction Rollback

Field	Size	Offset	Value	Comment
Command	2	0	94 00	
Data	<empty>			

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.60: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned

4.6 Export Commands

Log Messages of the TOE can be exported in two ways: filtered or unfiltered.

Doing a filtered export is an expensive operation that takes a very long time to complete. Therefore, it is recommended to always use a complete, unfiltered export (see Section 3.4) and delete the exported data afterwards (see Section 4.6.4) to effectively create an incremental export that runs at the fastest speed.

A filtered export is performed by first supplying the requested filter (see *Start Filtered Export*) and then repeatedly polling the filtered data with *Poll Filtered Export* until the export is complete. A filtered export can be aborted at any time with *Abort Filtered Export* and will automatically be aborted on power loss.

4.6.1 Start Filtered Export

Description

This command starts a filtered export of stored Log Messages by supplying a filter. The Log Messages are collected in the background and can be fetched by repeatedly calling *Poll Filtered Export*.

The exported Log Messages can be filtered based on their timestamp, transaction number, and the client that created the transaction. Filter criteria can be combined as defined in [BSI-TR-03153]. It is possible to filter based on

- *Transaction Number* and *Client-ID*
- *StartTransactionNumber* to *EndTransactionNumber* and *Client-ID*
- *TimeStampStart* to *TimeStampEnd* and *Client-ID*

All System and Audit Log Messages that were created between the first included Log Message belonging to a transaction start and the last included Log Message belonging to a transaction finish, will also be included in the exported data.

If the supplied filter is inconsistent, i.e. *Timestamp End* is lower than *Timestamp Start* or *Transaction Number End* is lower than *Transaction Number Start*, this command will fail with [`0x1007`: Invalid parameter].

Note: After issuing this command, all commands except of *Fetch Command Response*, *Poll Filtered Export*, and *Abort Filtered Export* are not allowed and will fail with [`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed]. This restriction does not apply to an unfiltered export.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.61: Command Data: Start Filtered Export

Field	Size	Offset	Value	Comment
Command	2	0	A0 00	
Timestamp Start	8	2		Timestamp as seconds since Unix Epoch. The timestamp will be interpreted as an unsigned number, which means only dates after 1970 are supported. If 0, it will be treated as the beginning of time. Big Endian.

Continued on next page

Table 4.61 – continued from previous page

Field	Size	Offset	Value	Comment
Timestamp End	8	10		Timestamp as seconds since Unix Epoch. The timestamp will be interpreted as an unsigned number, which means only dates after 1970 are supported. If `0xFFFFFFFFFFFFFFFF`, it will be treated as infinity. Big Endian.
Transaction Number Start	8	18		Start transaction number (inclusive). Big Endian.
Transaction Number End	8	26		End transaction number (inclusive). If `0xFFFFFFFFFFFFFFFF`, all transactions will be returned. If this is the same as *Transaction Number Start*, only transaction data belonging to this single transaction will be exported. Big Endian.
Client ID Length	1	34	L1	Maximum length is 30 bytes.
Client ID	L1	35		ASCII string representing the unique serial number of the client. Use a zero length string to not filter for a client ID.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.62: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x1007`: Invalid parameter

4.6.2 Poll Filtered Export

Description

After a filtered export has been initiated with *Start Filtered Export*, the actual data must be queried in small chunks by repeatedly calling this command. The returned data must be concatenated to form the final TAR archive.

The export is complete if this command returns a zero length chunk.

A filtered export is a very time consuming operation. The TSE will collect the data that matches the filter in the background and waits for the ERS to collect them. If the TOE did not find new matching data since the last call, the command will fail with [`0x2002`: Filtered Export: no new data, keep polling]. In that case, the ERS should repeat the command after a short delay to give the TOE some time to search for new data.

A filtered export either completely finishes by returning a zero length chunk, fails because of an error, or must be aborted with *Abort Filtered Export*. If the TSE loses power during a filtered export, the export will be aborted automatically and must be restarted from scratch.

If no data could be found that matches the supplied filter, this command will fail with [`0x2003`: Filtered Export: no matching entries, export would be empty].

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.63: Command Data: Poll Filtered Export

Field	Size	Offset	Value	Comment
Command	2	0	A1 00	

Response

In case of success, the SW is [`0x0000`: Execution successful] and the following additional data is returned, otherwise an error is returned (see next section).

Table 4.64: Response Data: Poll Filtered Export

Field	Size	Offset	Value	Comment
Export Data Length	2	0	R1	Length of the exported data in this response block. If this is `0`, the end of the exported data has been reached and the export is complete. Big Endian.
Export Data	R1	2		

Error Codes

Table 4.65: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x2001`: Filtered Export: no export in progress
`0x2002`: Filtered Export: no new data, keep polling
`0x2003`: Filtered Export: no matching entries, export would be empty

4.6.3 Abort Filtered Export

Description

Aborts a currently running filtered export. If no filtered export is in progress, the command also succeeds without errors.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.66: Command Data: Abort Filtered Export

Field	Size	Offset	Value	Comment
Command	2	0	A4 00	

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.67: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized

4.6.4 Delete Exported Data

Description

Deletes all data that has been successfully exported before.

This command requires a complete, unfiltered export and acknowledgement of the ERS (see Section 4.6.5) before data can be deleted.

No new data must have been generated since the last export in order to successfully execute this command.

Please note that after the *TSE Store* has been filled with more than 3gb of data, the next deletion might take up to 15 minutes, because the TSE runs an internal garbage collection to restore flash health and performance. This limitation does not apply if the deletion is performed with a *TSE Store* that is not filled with so much data.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the user *Admin* to be logged in (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.68: Command Data: Delete Exported Data

Field	Size	Offset	Value	Comment
Command	2	0	A2 00	

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.69: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned
`0x100F`: Not authorized
`0x1012`: Failed to delete, data not completely exported

4.6.5 Acknowledge Export

Description

After doing an unfiltered export, the host application can notify the TOE that it successfully received the exported data in order to allow execution of the *Delete Exported Data* command.

Permissions

- This command requires the state *CTSSInterfaceState* to be active (see Section 4.1).
- This command requires the state *TSEInitialized* to be active (see Section 4.1).

Command structure and parameters

Table 4.70: Command Data: Acknowledge Export

Field	Size	Offset	Value	Comment
Command	2	0	A3 00	
Export Size	8	2		Size of successfully received export data in bytes. Big Endian.

Response

The response does not carry any data except of the SW. In case of success, the SW is [`0x0000`: Execution successful], otherwise one of the following error codes is returned.

Error Codes

Table 4.71: Error Codes

`0x1001`: Unspecified, internal processing error
`0x1005`: Invalid command syntax
`0x1054`: Wrong state, self test must be run first
`0x1055`: Wrong state, passed self test required
`0x100D`: Wrong state, ongoing Data Import must be finished before this command is allowed
`0x1016`: Wrong state, ongoing Filtered Export must be finished before this command is allowed
`0x1053`: Wrong state, active CTSS interface required
`0x10FF`: Wrong state, TSE not initialized
`0x10FE`: Wrong state, TSE decomissioned

CHAPTER

FIVE

APPLICATION NOTES

This chapter contains information that did not fit into the previous chapters. It is meant to be a loose collection of information that is important for the use of the TOE.

5.1 Usage of TSE in RAW access

For simple architectures and embedded applications without a file system implementation the host can access the special TOE files in raw mode, which means it uses the sector offsets of the files and directly reads from and writes to these sectors on the raw device instead of going through the file system.

How RAW access is implemented on the client architecture goes beyond the scope of this document. As an example: if one would implement RAW access on a Linux system, it would mean to write to or read from the block device (e.g. `/dev/sdc`) directly instead of interacting with the mounted file system.

The logical block addresses (LBAs) of the TOE special files are announced in sector 8 of the TOE device:

Table 5.1: RAW Access Info Table

Field	Size	Offset	Comment
Logical Sector address of *TSE_INFO.DAT*	4	0	LBA of *TSE_INFO.DAT*
Logical Sector address of *TSE_COMM.DAT*	4	4	LBA of *TSE_COMM.DAT*
Logical Sector address of *TSE Store* begin	4	8	LBA of *TSE Store* begin
RFU	500	12	RFU (all 00)

5.2 Formatting the TSE

The security mechanism inside the TOE requires a fixed partition structure. Therefore, formatting the TOE is prevented and will fail.

Please note that while the partitioning and format of the partition is protected, there are some critical structures on the file system that must be writable by the host in order to allow creating new files. That means that if these structures are overwritten with invalid data (e.g. by overwriting the whole device with zeros), the TOE might not be mountable anymore on the host system.

While this temporarily prevents accessing the TOE, this situation can be fixed by the host by restoring the file system structure. Swissbit provides a tool upon request to do this for Windows and Linux. After restoring the file system structure, all previously stored transactions can be read again (i.e. there is no possibility for a permanent loss of transaction data), but files that were created in the freely writable user space might be lost.

5.3 Number of possible transactions and size of standard partition

The size of the *TSE Store* is fixed to 6.5 GB and suits approximately 3.5 million transactions each with 512 bytes process data in average and 2 signatures applied - one at Transaction Start and one at Transaction Finish (i.e. each transaction consists of roughly 4 blocks). To store more transactions, export the data and delete the *TSE Store* afterwards (see Section 4.6.4).

5.4 Amount of possible signatures

Due to technical limitations the number of digital signatures is typically limited, but it is guaranteed to be at least 20 million signatures. Beyond this, it may still be possible to create further signatures. In case the internal limit of signatures of the builtin security module has been reached the potential risk for a loss of data must be taken into account by the user, since access to data might require the ability to perform signatures.

5.5 TSE full detection

The remaining number of possible transactions can be calculated by **TSE Capacity** – **TSE Current Size** (please refer to *TSE Status*).

Please note that in case the *TSE Store* does not have enough free space to store the Log Message of an operation, the command that triggers the Log Message will fail with [`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*]. However, the effect of the command will still be applied. For example, calling *Login User* will fail, but the user will in fact be logged in afterwards. This is done to ensure that a user can still be logged in even if the *TSE Store* is full, which is a requirement for actually being able to delete the stored data in order to reclaim free space in the *TSE Store*. In that case the Log Message belonging to the operation will be lost and there will be a gap in the signature counter of Log Messages if the TSE is later being brought back into a fully operational state by deleting the stored data. Since error code [`0x1017`: Operation failed, not enough remaining capacity in *TSE Store*] can be raised for both, successful and unsuccessful operations, it is advised to monitor the remaining free space of the *TSE Store* and export and delete the stored data in case the remaining space gets low.

Data Imports that are too big to fit into the remaining space will simply be rejected before any signature is created to make sure that no Log Message belonging to a transaction gets lost.

5.6 Host File System Caching Considerations

Reading the response in *TSE_COMM.DAT* may fail if file caching mechanisms apply. Please ensure the file is opened while the file system cache is deactivated or bypassed. The read back worked if the data structure as described in Section 4 is retrieved, especially the *Write Index* in the response is increased per write on *TSE_COMM.DAT*. By verifying the increment of the *Write Index* the host can verify that a write and respective read of the response has happened.

A similar restriction applies when reading the stored transaction directly from the TOE as described in Section 3.4. Reading the data is only possible if the file system cache is deactivated, bypassed, or not populated yet, e.g. because the TOE was just inserted into the host system and thus the file system cache of the device is empty.

5.7 Driver support for PC systems

For simple transactions on PC systems Swissbit provides an easy to use library for Linux and Windows. Please contact your sales representative for further information.

5.8 Initial PUK and PINs

Each TSE will have a different value for the *Admin* PIN and PUK and the *TimeAdmin* PIN. The initial values are derived from the *TSE Serial Number* (see *TSE Status*) and a customer (customer refer to the developer of the ERS in this context) specific *Seed* and are then mapped to the ASCII characters '0' - '9' (i.e. they only consist of

digits). The character set of new PINs and PUKs are not restricted, but it is recommended to stick to alphanumeric values to ensure they can be entered on all used systems without issues.

The algorithm used to derive the initial values is implemented as part of the TSE SDK and works like this:

1. Input: `tseSerialNumber`, `seed`
2. Calculate SHA-256 of concatenated `seed` and `tseSerialNumber`: `hash = SHA256(seed | tseSerialNumber)`
3. Take the first 24 bytes of `hash` and split it into three 64 bit unsigned integers (parsed using Big Endian notation).
4. For the *Admin PUK*, take the first number and take it modulo by 1.000.000. Take the remainder as PUK.
5. For the *Admin PIN*, take the second number and take it modulo by 100.000. Take the remainder as PIN.
6. For the *TimeAdmin PIN*, take the third number and take it modulo by 100.000. Take the remainder as PIN.
7. The PUK will now be a number between 0 and 999.999 and the PINs will be numbers between 0 and 99.999.
8. To form the final values, convert the numbers to string while padding them to the required length of 6 (for PUK) and 5 (for PINs) with ASCII '0's.

Note: The seed for development samples is `SwissbitSwissbit`.

5.9 TSE Setup

Figure Fig. 5.1 (which is identical to Figure 4 in [AGD]) shows all necessary steps that bring the TOE from the clean factory state into a usable state. Afterwards, the time can be synchronized and the TOE is ready to receive transactions.

It is not necessary to register each ERS at the TOE before initializing it. Clients can be registered later at any time as long as the *Admin* user is logged in.

5.10 TSE Usage for Transactions

Fig. 5.2 shows how to use the TOE in daily operation to perform transactions, starting from powering on the TOE.

5.11 Exceptional Error Cases

In case the TOE encounters an invalid state (while not explicitly executing the self test as part of command *Run Self Test*), it will reboot itself. This is done to ensure the TOE re-enters a valid state. The ERS can detect this situation by checking if the *Write Index* of the command response was reset to 0 (see Section 3.5).

These exceptional error cases are not expected to be run into and are most likely indicating a hardware fault. All regular error cases are handled by error codes as outlined in the previous chapters.

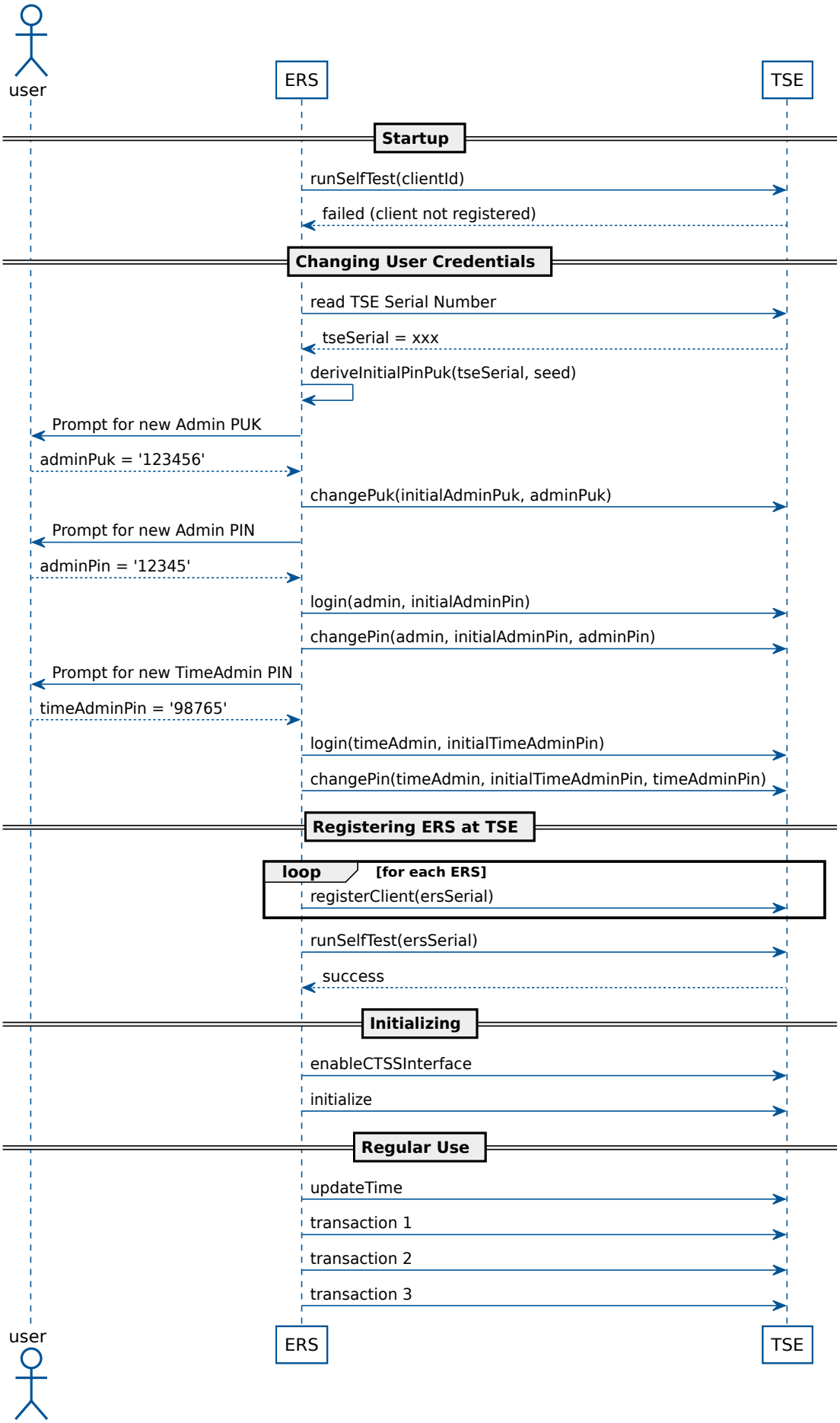

Fig. 5.1: TSE Setup

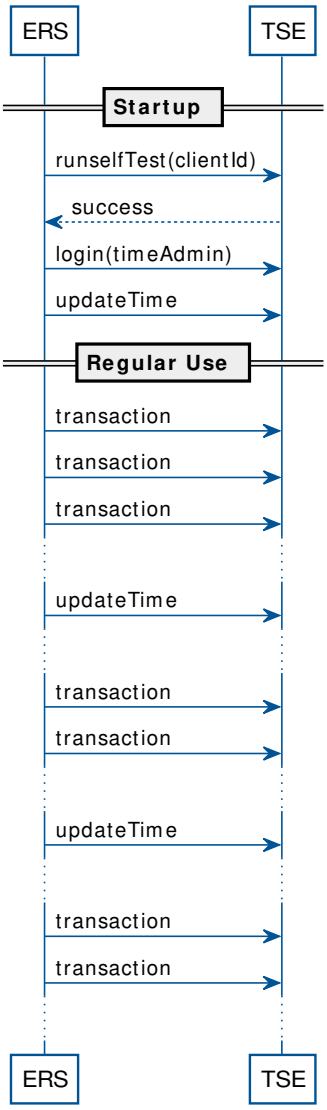

Fig. 5.2: TOE Usage for Transactions

5.12 Mapping of SFR

In order to ensure that all SFR from [ST] are sufficiently covered by commands of the interface of the TOE, the following table summarizes the mapping.

Table 5.2: Mapping of SFR to commands

SFR	command enforcing	command supporting	Comment
FDP_ACC.1/LM	• all commands		as every command of the TOE implements access control, one can argue that all commands support this SFR.
FDP_ACC.1/UCP	• *TSE Firmware Update Transfer* • *TSE Firmware Update Apply*		The access control policy for the update of the code package is implemented by *TSE Firmware Update Transfer* and *TSE Firmware Update Apply*.
FDP_ACF.1/LM	all commands	as every command of the TOE implements access control, one can argue that all commands support this SFR. The list of rules that are defined in FDP_ACF.1.2,2,3/LM is specifically implemented by *Data Import Initialize*, *Data Import Finalize*, *Start Filtered Export*, *Acknowledge Export*, *Update Time*, *Delete Exported Data*, *Register Client* and the *File Based Interface*	The access

Continued on next page

Table 5.2 – continued from previous page

SFR	command enforcing	command supporting	Comment
FDP_ACF.1/UCP	• *TSE Firmware Update Transfer* • *TSE Firmware Update Apply*		The functionality for firmware update is exposed via the commands *TSE Firmware Update Transfer* and *TSE Firmware Update Apply*.
FDP_ETC.2/DTBS			The export of data to the CSP is a purely internal functionality of the TOE. For this reason, there are no commands that implement this functionality.
FDP_ETC.2/LM	• *File Based Interface* • *Start Filtered Export* • *Poll Filtered Export* • *Delete Exported Data* • *Acknowledge Export*		The functionality for export of log messages as described in FDP_ETC.2/LM is implemented by the commands that provide the functionality to export the log messages. Also the file based interface is an implementation of this functionality.
FDP_ITC.2/TD	• *Data Import Initialize* • *Data Import Transfer* • *Data Import Finalize*	• *Register Client*	The functionality as required by FDP_ITC.2/TD is implemented by the commands that can be used to store transactions with the TOE. Also, the command to register a client supports this functionality as it is used to submit the information of the allowed ERS to the TOE.

Continued on next page

Table 5.2 – continued from previous page

SFR	command enforcing	command supporting	Comment
FDP_ITC.2/TSS	• *Data Import Initialize* • *Data Import Transfer* • *Data Import Finalize*		The functionality described in FDP_ITC.2/TSS is an internal functionality that concerns the interface between the TOE and the CSP and is not directly exposed to the user. For this reason, it is not directly exposed via a command. One can however argue that the functionality to start, update and finish transactions are an implementation of this functionality.
FDP_ITC.2/UCP	• *TSE Firmware Update Transfer* • *TSE Firmware Update Apply*		The functionality for firmware update is exposed via the commands *TSE Firmware Update Transfer* and *TSE Firmware Update Apply*.
FDP_RIP.1/UCP	• *TSE Firmware Update Transfer* • *TSE Firmware Update Apply*		The functionality as required by FDP_RIP.1/UCP is implemented by commands commands *TSE Firmware Update Transfer* and *TSE Firmware Update Apply*.
FIA_AFL.1	• *Login User* • *Unblock User*		The commands *Login User* and *Unblock User* implement the authentication failure handling as required by FIA_AFL.1.
FIA_ATD.1	• *Login User*		The attributes Identity, Authentication Reference Data and Role that have to be maintained for the roles Admin and Time Admin are implemented by the command *Login User*. The attributes SerialNumber belonging to the ERS and PACE-PIN belonging to the CSP are not exposed via an interface.

Continued on next page

Table 5.2 – continued from previous page

SFR	command enforcing	command supporting	Comment
FIA_UAU.1	• *Login User*	all commands	While the actual authentication of a user is implemented by the command *Login User*, each command of the TOE implements the functionality of FIA_UAU.1 in so far that it ensures that the correct authentication state is existing.
FIA_UAU.5	• *Login User*	all commands	The password authentication that is required for the Admin and Time Admin role is implemented by the *Login User* command. Further, all commands of the TOE enforce the correct authentication state. The successful PACE-Channel establishment for the CSP role and the provision of valid ERS Serial Number for the CTSS Interface role are only exposed in case of an error. For this reason all commands are rated SFR-supporting for this SFR.
FIA_UAU.6	• *Login User*		The password authentication as described in *Login User* only sets the status of the user to the next reboot. This implements FIA_UAU.6.
FIA_UID.1	• *Login User* • *Run Self Test*		• The initial role of the user is assumed to be unidentified user • The command *Run Self Test* is the only command that can be run without user identification • other roles are associated with the user only after successful authentication via *Login User*
FIA_USB.1	• *Login User* • *Run Self Test*		• The initial role of the user is assumed to be unidentified user • The command *Run Self Test* is the only command that can be run without user identification • The other roles are associated with the user after successful authentication via *Login User*
FMT_MOF.1	• *Register Client*		The rules as described in FMT_MOF.1 are enforced as follows: - (1) Enabling and disabling of password authentication is not implemented and does not require any command. - (2) The TOE does no provide a method to determine the life time limit of open transactions. - (3) The serial number that can be set by the Administrator by the use of the command *Register Client* is used during self testing. This is the only information that can be configured for the self test - (4) The TOE does not allow to change any other information - (5) The TOE does not allow to change any other information

Continued on next page

Table 5.2 – continued from previous page

SFR	command enforcing	command supporting	Comment
FMT_MSA.1			The TOE does not offer any functionality to change the default values as described in FMT_MSA.1 and implements the SFR this way. For this reason, no commands are used to implement this SFR.
FMT_MSA.2	• *Register Client* • *Deregister Client*		The TOE implements the functionality as required by FMT_MSA.2. However, the transaction counter is stored internally and can (for good reason) not be administered. The only functionality that is visible at the TSFI is the definition of accepted values for the *serial number of the ERS* as require by FMT_MSA.2.
FMT_MSA.3	• none	• none	As described in the [ST], there is no interface to configure default values for security attributes, which implements FMT_MSA.3.2. By sticking to the provided default values from [PP-SMAERS], restrictive choices were made to implement FMT_MSA.3.1.
FMT_MSA.4	• *Data Import Initialize* • *Data Import Transfer* • *Data Import Finalize*		The attributes described in FMT_MSA.4 are kept only internally within the TOE. For this reason, no command can be mapped here. One could argue that the commands to start, update and finish transactions implement this SFR.
FMT_MTD.1/AD	• *Change PUK* • *Change PIN*		The role Administrator can change the Administrator PIN, the role TimeAdmin can change the TimeAdmin PIN. In addition, Administrator can change the Administrator PIN, TimeAdmin PIN and PUK using the PUK as credential.
FMT_MTD.3/PW	• *Change PUK* • *Change PIN*		The initial PINs are derived from the TOEs serial number and stored at production time. They have to be changed after the first login. The file TSE_INFO.DAT in the file system indicates, if the PINs were already changed.
FMT_SMF.1	• *Register Client* • *Deregister Client*		The management of acceptable ERS Serial Numbers is implemented in command *Register Client*. Please refer directly to the description of the other SFRs in this table for the rest of this SFR.

Continued on next page

Table 5.2 – continued from previous page

SFR	command enforcing	command supporting	Comment
FMT_SMR.1	*Login User*	• *Unblock User* • *Logout User* • *Unblock User* • *Change PUK* • *Change PIN*	
FPT_TDC.1	*Export Commands*		With respect to the formats of im- and exported data, the TOE is conformant to the required specifications.
FPT_FLS.1	none	*Run Self Test*	The commands that expose the tests support this command but the functionality that enters the secure state is purely internal.
FPT_TEE.1	*Run Self Test*		*Self Test Commands* implements the self test as required by FPT_TEE.1
FPT_TST.1	*Run Self Test*	none	
FCS_CKM.1			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FCS_CKM.4			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FCS_COP.1			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FCS_RNG.1			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FIA_API.1			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FIA_UAU.5/TC			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.
FIA_ITC.1/TC			The functionality from this SFR addresses the Trusted Channel between the TOE and the CSP. This functionality is not described in this document.

BIBLIOGRAPHY

[FAT32] Microsoft Extensible Firmware Initiative FAT32 File System Specification, Version 1.03, 2000

[USB-Spec] Universal Serial Bus Specification , Revision 2.0, April 2000

[SD-Spec] SD Specifications, Part 1, Physical Layer - simplified Specification, Version 5.00, August 2010

[BSI-TR-03153] Technische Richtlinie BSI TR-03153 Technische Sicherheitseinrichtung für elektronische Aufzeichnungssysteme, TR-03153, Version 1.0.1

[BSI-TR-03151] Technical Guideline BSI TR-03151 Secure Element API (SE API), TR-03151, Version 1.0.1

[BSI-TR-03111] Technical Guideline BSI TR-03111 Elliptic Curve Cryptography, TR-03111, Version 2.10

[PP-SMAERS] Common Criteria Protection, Profile Security Module Application for Electronic Record-keeping Systems, Aktuell in Version 0.7.2

[PPC-CSP-TS-Au] Common Criteria Protection Profile Configuration, Cryptographic Service Provider - Time Stamp Service and Audit, Version 0.9.5

[AGD] swissbit TSE - Guidance Manual, Version 1.0.0

[ST] Security Target Swissbit TSE SMAERS