
# STS Association

## STS 600-8-6

**Edition 1.6**

**March 2022**

**Standard Transfer Specification – Key management – Hardware security module API**

CONTENTS

**Revision History**

| Edition | Clause | Date | Change details from previous edition |
|---|---|---|---|
| 1.0 | | August 2017 | Initial Revision - based on PR-D2-0970 Rev1.14 2016 document published by Prism Payment Technologies |
| 1.1 | 5.5.2 | August 2017 | Changed reference to STS600-4-2 Ed1.3. Removed Ed number. |
| 1.2 | 6.2.2.3 | Sept 2017 | Editorial - incorrect reference cited. Changed reference to STS600-4-2 Ed 1.1 from ED1.3. |
| 1.3 | 7.6.1 | June 2018 | Made this clause informative. |
| 1.4 | 7.5.1.1 | Sept 2018 | Included Class2, Subclass10 extended token set (no longer Reserved).<br>Added reference to STS202-5. Removed references to WG8 document, STS202-1 , STS202-2, STS202-3, COP402, STS501-8. |
| | general | | Various editorial changes |
| | Appendix F | | Added Permission ERROR code 98 to error table. |
| | general | | Reformatted document to remove hanging clauses and correct table header fonts. Removed brackets "[]" from cited references. |
| 1.5 | general | Jan 2019 | Reformatted all examples |
| | 7.4 | | Made the Vending Authorisation commands SM?CQ and SM?CI proprietary since they do not form part of the STS requirements. |
| | Table 6 | | CLM incorrectly represented as 232 instead of 232 |
| | general | | General editorial updates. |
| | 8.2 | | TDES should be DES. Updated descriptions for KCV calculations |
| | Table 6 | | Made the presence of the SGN attribute optional in the SM?GA and SM?KL commands. |
| 1.5 draft 2.0 | | 17/02/2019 | Editorial review for compliance with new STSA standards template and harmonization with STS 600-4-2, STS 600-10-1 and IEC 62055-41 |

| | | | |
|---|---|---|---|
| 1.5 draft 2.1 | | 13/03/2019 | Technical and editorial corrections after review with Franco<br>Circulate to WG8 for comments |
| 1.5 draft 2.2 | | 16/03/2019 | Reset all data types back to original prior draft 2.0 as per Trevor's comments<br>Add communication interface reference model<br>Circulate to WG8 for comments |
| 1.5 draft 2.3 | | 21/04/2019 | Resolution of comments on draft 2.2 |
| 1.5 draft 2.4 | | 06/07/2019 | Reformat in STS template |
| 1.5 draft 2.4 | | 05/08/2019 | Circulate to WG8 as draft CDV for editorial comments |
| 1.5 | | 20/09/2021 | published |
| 1.6 | 6.7, Annex G | Mar 2022 | Added vending authorisation commands<br>Removed legacy SM?ID command, and the implied limitations on MID and FWID representation (compared to their STS600-4-2 definitions)<br>Clarified that EA07 DITKs are not limited to Odd parity (for backwards compatibility) |
| | 6.6.6.1 | | Added DDTK to the possible keychange tokens generated |
| | 6.6.6.3 | | Changed SM?SV to SM?VT |

STANDARD TRANSFER SPECIFICATION ASSOCIATION

_____

# STANDARD TRANSFER SPECIFICATION –

## Key management – Hardware security module API

## FOREWORD

1) The Standard Transfer Specification Association (STSA) is a worldwide organization for standardization comprising all members of STSA. The object of STSA is to develop, maintain and promote international use of the Standard Transfer Specification (STS). To this end and in addition to other activities, STSA publishes Standards, Technical Specifications, Technical Reports, Codes of Practice and Guides (hereafter referred to as "STSA Publication(s)"). Their preparation is entrusted to technical working groups; any STSA member interested in the subject dealt with may participate in this preparatory work. STSA collaborates closely with the International Electrotechnical Commission (IEC) in accordance with conditions determined by agreement between the two organizations. As such STSA performs the role of Registration Authority of IEC 62055-41, IEC 62055-51 and IEC 62055-52 on behalf of IEC.
2) The formal decisions or agreements of STSA on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each working group has representation from all interested STSA members.
3) STSA Publications have the form of recommendations for international use and are accepted by STSA Board of Directors in that sense. While all reasonable efforts are made to ensure that the technical content of STSA Publications is accurate, STSA cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
5) STSA provides attestation of conformity. Independent testing bodies provide conformity assessment services and recommendations to STSA Board of Directors who provides conformance certificates and access to STSA marks of conformity.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to STSA or its directors, employees, servants or agents including individual experts and members of its technical working groups for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this STSA Publication or any other STSA Publications.
8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this STSA Publication may be the subject of patent rights. STSA shall not be held responsible for identifying any or all such patent rights.

Standard Transfer Specification STS 600-8-6 has been prepared by working group 8.

This 1.5th edition cancels and replaces the 1.4th edition published in September 2018. This edition constitutes a technical and editorial revision.

The editorial revision constitutes alignment with the new STSA standards template and harmonization with STS 600-4-2, STS 600-10-1 and IEC 62055-41.

This edition includes the following significant technical changes with respect to the previous edition:

a) Removed the proprietary vending authorisation commands SM?CQ and SM?CI;
b) CLM correction in Table F.1;
c) Changed TDES to DES in A.2.1;
d) Made the presence of the SGN attribute optional in the SM?GA and SM?KL commands.

The text of this standard is based on the following documents:

| FDS | Report on voting |
|---|---|
| STS600-8-6 Ed1.5 | STS600-8-6RVD_03_22 |

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with STSA Directive STS 2100-1.

# INTRODUCTION

The Standard Transfer Specification (STS) is a secure message protocol that allows information to be carried between point of sale (POS) equipment and payment meters and it caters for several message types such as credit, configuration control, display and test instructions. It further specifies devices and codes of practice that allows for the secure management (generation, storage, retrieval and transportation) of cryptographic keys used within the system.

Hardware security modules (HSMs) have been in use since the inception of the STS. The STS Association acquired the intellectual property rights to the application programming interface (API) for these HSMs in 2017. This standard represents the conversion of the original proprietary specifications into an open STS standard.

This standard is intended for use by users of HSMs for the generation of tokens that comply with the STS. Additional specifications and operating modes may apply to HSMs other than those contained in this standard, please refer to the suppliers manuals supplied with the HSM for such additional commands and operating modes.

It is also intended to be used by developers and manufacturers of HSM devices for deployment into STS eco-systems.

The API specified in this standard supports the STS 600-4-2 key management standard, which has resulted in a number of enhancements and differences to the legacy STS firmware API and key management practices:

- support for TID rollover, DKGA04, EA11, and STS600 key management;
- support for currency transfer tokens;.
- initialisation is done at the HSM manufacturer’s premises. The HSM does not need to be physically sent to the KMC as was the case with the legacy STS HSMs. This simplifies the logistics of procuring an HSM from the HSM manufacturer when compared to a legacy HSM;
- a new KeyLoadFile (KLF) format has been defined (in STS600-4-2). This means that an HSM cannot use legacy KLFs or be key injected by a legacy KMC. The HSM must request a KLF by generating a challenge using SM?KC (see 6.3.3) that is sent to the KMC. The KMC uses the challenge to prepare the KLF, which contains a response-to-challenge and the VendingKey instances (VKs) that the HSM is authorised to use. The KLF is sent to the vending system where the response is introduced to the HSM using SM?KR (see 6.3.4). Individual VKs are loaded using SM?KL (see 6.3.5);
- the new KLF format allows for attributes to be cryptographically bound to the VK. These attributes constrain key use, allowing the supply group owner to specify risk management rules that are enforced by the HSM. In particular, these rules can protect against unauthorised vending by enforcing unit/currency vending limits (per SGC + KRN pair), and enforcing key expiry (based on TokenID and real time clock (RTC);
- when issuing tokens, a TokenID-to-RTC window ensures that the TokenID is “close to” the HSM’s real-time clock, preventing accidental (or deliberate) vending of old or wildly wrong tokens;
- the following features are not required in the STS eco-system, so they have been removed to improve security: message encryption, decryption and authentication; password/data storage; and key export. Key export (legacy command SM?FK) would introduce a vulnerability in vending limit control, allowing risk management protections to be subverted;
- the new API commands employ a consistent message and field encoding called “Printable Type-Value-Delimiter” (PTVD). A consistent encoding facilitates safe handling of data, and makes implementation easier (see 5.4);

- to comply with international HSM security requirements for key component entry, the dispenser ROM key (DITK) components are loaded by the meter manufacturer via a dedicated serial port. This functionality is only permitted on HSMs that have a dedicated port for key component entry.

# STANDARD TRANSFER SPECIFICATION –

# Key management – Hardware security module API

## 1 Scope

This standard specifies a set of service request and service response message pairs for data exchange between a client application process and the HSM application process.

It does not specify the lower layers of the communications interface protocol stack, which is left to the HSM manufacturer to implement in accordance with market requirements.

All commands apply to both vending HSM devices and manufacturing HSM devices, except where indicated otherwise.

## 2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62051, *Electricity metering – Glossary of terms*

IEC 62055-31:2005, *Electricity metering – Payment systems – Part 31: Particular requirements – Static payment meters for active energy (classes 1 and 2)*

IEC 62055-41:2014, *ELECTRICITY METERING – PAYMENT SYSTEMS – Part 41: Standard transfer specification (STS) – Application layer protocol for one-way token carrier systems*

IEC 62055-51:2007, *Electricity metering – Payment systems – Part 51: Standard transfer specification (STS) – Physical layer protocol for one-way numeric and magnetic card token carriers*

STS 600-4-2: Ed 1.2, *STANDARD TRANSFER SPECIFICATION – Companion Specification – Key Management System*

STS 600-10-1: Ed1.0, *Standard Transfer Specification – Key Management – Requirements for hardware security modules*

STS 202-5: Ed1.0, *Addendum to IEC62055 - 41: Payment Systems - Standard Transfer Specification (STS) - Class 2 token extensions*

## 3 Terms, definitions and abbreviations

### 3.1 General

For the purposes of this standard, the terms and definitions given in IEC 62051, IEC 62055-31, IEC 62055-41, IEC 62055-51, STS600-4-2 and the following terms apply.

Where there is a difference between the definitions in this standard and those contained in other referenced standards, then those defined in this standard shall take precedence.

### 3.2 Definitions

| Term | Definition |
|---|---|
| WRAPPED-KEY | See 7.5 in STS 600-4-2 |
| $PUBKEY_{KMC}$ | See 7.2 in STS 600-4-2 |

| | |
|---|---|
| TIMESTAMP | See 5.1.4 in STS 600-4-2 |
| VKLOADREQSM | See 7.3 in STS 600-4-2 |
| Literal “XXX” | A specified content string of printable ASCII characters |
| Number | See 5.4 for PTVD field types |
| PrintableString | See 5.4 for PTVD field types |
| DateTime | See 5.4 for PTVD field types |
| <CR> | ASCII carriage return character |
| HexOctets | See 5.4 for PTVD field types |
| Card format | See 7.5.1 in STS 600-4-2 |
| IDENT | See 5.1.3 in STS 600-4-2 |

## 3.3 Abbreviated terms

| Term | Definition |
|---|---|
| API | Application Programming Interface |
| CSP | Critical Security Parameter (keys and other sensitive data stored in the HSM) |
| DKGA | DecoderKeyGenerationAlgorithm |
| HLSM | High Level Security Module |
| HSM | HardwareSecurityModule, a device used to protect VKs and to create STS tokens |
| KCED | KeyComponentEntryDevice |
| KCV | KeyCheckValue |
| KEK | KeyEncryptionKey, a digital key shared between the KMC and the HSM that is used to protect VKs in transit (in the KLF) |
| KEN | KeyExpiryNumber |
| KLF | KeyLoadFile, a data file used to transport VKs from the KMC to an HSM |
| KMC | KeyManagementCentre, the infrastructure that: manages SGs & SGCs, creates VKs, and distributes VKs to HSMs (in a KLF, protected by a KEK). |
| KRN | KeyRevisionNumber |
| MSE | Meter-specific Engineering token |
| NONCE | NumberUsedOnce |
| PTVD | PrintableTypeValueDelimiter |
| RTC | RealTimeClock |
| SG | SupplyGroup, a group of STS prepayment meters that are linked to the same VK |
| SGC | SupplyGroupCode, identifies a supply group; SG and SGC are often used interchangeably |
| STS | Standard Transfer Specification |
| TID | TokenIdentifier |
| VK | VendingKey, a digital key associated with an SGC and which is used by an HSM to create STS tokens |

## 3.4 Notation and terminology

Throughout this document the following rules are observed regarding the naming of terms:

- entity names, data element names, function names and process names are treated as generic object classes and are given names in terms of phrases in which the words are capitalized and joined without spaces. Examples are: SupplyGroupCode as a data

element name, EncryptionAlgorithm07 as a function name and TransferCredit as a process name (see note);

- direct (specific) reference to a named class of object uses the capitalized form, while general (non-specific) reference uses the conventional text i.e. lower case form with spaces. An example of a direct reference is: “SupplyGroupCode is linked to a group of meters”, while an example of a general reference is: “A supply group code links to a group of meters”;
- other terms use the generally accepted abbreviated forms like PSTN for Public Switched Telephone Network.

NOTE The notation used for naming of objects has been aligned with the so called “camel-notation” used in the common information model (CIM) standards prepared by IEC TC 57, in order to facilitate future harmonization and integration of payment system standards with the CIM standards.

### 3.5 Numbering conventions

In this document, the representation of numbers in binary strings uses the convention that the least significant bit is to the right, and the most significant bit is to the left.

Numbering of bit positions start with bit position 0, which corresponds to the least significant bit of a binary number.

Numbers are generally in decimal format, unless otherwise indicated. Any digit without an indicator signifies decimal format.

Binary digit values range from 0 to 1.

Decimal digit values range from 0 to 9.

Hexadecimal digit values range from 0 to 9, A to F and are indicated by a preceding 0x

ASCII characters are indicated by encapsulating quotation marks “”

An octet string as a sequence of Base16 digits is indicated by x’ (see 3.3 of STS 600-4-2)

## 4 Core concepts

### 4.1 Essential knowledge

This API standard assumes that the reader has reasonable knowledge of the STS ecosystem and, in particular, should be familiar with:

- IEC 62055-41 as the core STS standard, which defines tokens for the transfer of credit and other instructions between a point of sale and a meter;
- The key management process described in STS 600-4-2, and its associated data formats;
- STS 600-4-2 defines the key management infrastructure and associated processes that manage cryptographic keys.

The reader should also be conversant with the terms, definitions and abbreviations widely used in STS standards as given in clause 3.

### 4.2 Identification

All STS compliant HSMs use the following identifiers as given in Table 1

**Table 1: HSM identifiers**

| Identifier | Description | Reference |
|---|---|---|
| Manufacturer | The manufacturer of the HSM | 7.1 of STS 600-4-2 |
| MID | The module identifier of the HSM | 7.1 of STS 600-4-2 |
| GNT | The time stamp at which the HSM public/private key-pair was generated | 7.1 of STS 600-4-2 |
| Fingerprint | A hash that binds preceding fields and the record type to the HSM public key | 7.1 of STS 600-4-2 |
| HWID | The hardware model and revision identifier of the HSM | 7.3 of STS 600-4-2 |
| FWID | The firmware application and version identifier of the HSM. | 7.3 of STS 600-4-2 |
| IDSM | Unique identifier of the public key of the HSM | 7.3 of STS 600-4-2 |
| FirmwareHash | A unique identifier for the firmware build of the HSM | 0 |

These identifiers may be queried using the following commands:

- SM?DI (see 6.2.1);
- SM?QI (see 6.2.2);

### 4.3 Firmware type command groupings

HSM functionality is determined by the firmware type, as shown in Table 2:

**Table 2: API command groups supported by firmware type**

| Available command groups | Vending firmware | Manufacturing firmware |
|---|---|---|
| General and diagnostic commands (see 6.2) | Yes | Yes |
| Operational key management commands (see 6.3) | Yes | Yes |
| STS transfer credit commands (see 6.4) | Yes | No |
| STS management commands (see 6.5) | Yes | Yes |
| STS manufacturing commands (see 6.6) | No | Yes |

Vending firmware is used by a token vending system to create STS tokens.

Manufacturing firmware is used in a manufacturing environment to introduce keys to meters. It can issue management tokens but not credit tokens, and includes a special command to change a meter from a dispenser ROM key (DITK) to a default/common/unique key. To comply with international hardware HSM security requirements for key component entry, the dispenser ROM key components are loaded by the meter manufacturer via a dedicated serial port that is physically separate from the command interface port. Manufacturing commands are only available on HSMs with manufacturing firmware and that have a dedicated port for key component entry.

The HSM may support additional commands, including proprietary commands necessary for the production and maintenance of the HSM. Such commands are out of scope for this standard.

### 4.4 Communicating with the HSM

The communication interface reference model is given in Figure 1.

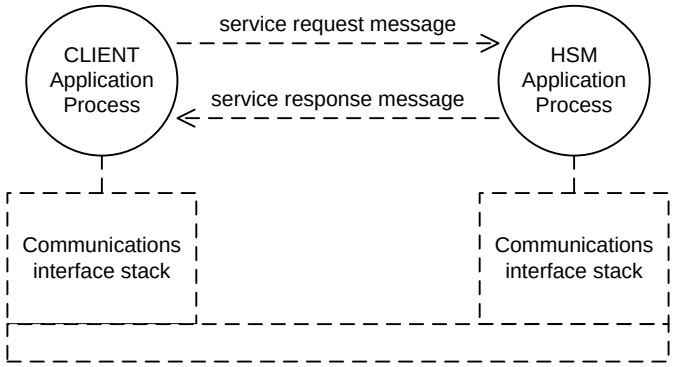


**Figure 1 – Communication interface reference model**

Data exchange takes place between the client application process and the HSM application process by means of service request and service response message pairs as given in clauses 5 and 0.

The protocol is completely independent of any communication interface stack, which is left to the HSM manufacturer to implement in accordance with market requirements.

### 4.5 Key management

#### 4.5.1 Key management session

To operate in an STS environment an HSM requires VendingKey instances (VKs), which it must receive from a key management centre (KMC). To this end the HSM implements the key management process as defined in STS600-4-2:

1. Get a KMC public key ($PUBKEY_{KMC}$) from the KMC.

2. Start a key management session by generating a VK load request (VKLOADREQ):

    - The HSM command SM?KC (see 6.3.3) is used in this step, and requires the $PUBKEY_{KMC}$. Key management state is stored in a KEK slot associated with the KMC's public key. If there is already a slot associated with the public key it is reused; otherwise a free slot is allocated.

3. Send the VKLOADREQ to the KMC. The KMC generates a KLF (see 4.5.2) that contains a VK load response (VKLOADRESP) and one or more VendingKey instances (each as a WRAPPED-KEY record).

4. Receive the KLF from the KMC and load the VKs:

    - The HSM command SM?KR (see 6.3.4) is used to load the VKLOADRESP, which establishes a key encryption key (KEK) between the KMC and the HSM. A successful SM?KR deletes from the HSM all VKs associated with the KEK slot, in preparation for loading new VKs from the KLF. In HSMs with vending firmware, all token vending (of credit and MSE tokens) is inhibited for VKs associated with the KEK slot, until a SM?KF command is issued.

    - Each VK can be loaded using the HSM command SM?KL (see 6.3.5), which accepts a WRAPPED-KEY protected by the KEK, unwraps it, and stores the VK in a register in the HSM.

To facilitate risk management in HSMs with vending firmware, this key management process is extended to include a final step:

5. Finish the key loading session and return to normal operation:

    - The HSM command SM?KF (see 6.3.6) informs the HSM that key loading is complete. The KEK slot is locked and no further VKs can be loaded (using

SM?KL) in this key management session. In HSMs with vending firmware the token vending inhibition is lifted, allowing credit and meter-specific engineering (MSE) tokens to be generated.

- It is not necessary to issue an SM?KF to HSMs with manufacturing firmware. The STS manufacturing commands are not inhibited by the key loading process, so the key load session can be kept open indefinitely, allowing VKs to be loaded (SM?KL) or deleted (SM?KD) as required.

### 4.5.2 KeyLoadFile(KLF)

A KLF is an STS600-4-2 file-of-records containing a VKLOADRESP and zero or more WRAPPED-KEY records. Its construction is given in clause 12 of STS 600-4-2 A KLF is received from the KMC during a key management session (see 4.5.1).

### 4.5.3 KeyEncryptionKey (KEK) slots

During a key management session a KEK is established between the HSM and a KMC. The KEK is stored in a “KEK slot” (also called a “KMC slot”), and that slot is assigned to the KEK from the start of the key management session. Each HSM has a number of KEK slots and can thus receive VKs from multiple KMCs. Each KEK slot is identified by the KMC’s name (specifically: the name is the MID part) of the subject of the KMC’s $PUBKEY_{KMC}$, as defined in 7.2 of STS600-4-2). Use SM?GL (see 6.3.1) to retrieve a list of KEK slot identifiers.

### 4.5.4 KEK slot state machine

The order of HSM commands in a key loading session is enforced by a state machine in the KEK slot, which is described in Table 3.

**Table 3: KEK slot state machine**

| KEK slot current state | KEK slot next state, after indicated command | | | | |
|---|---|---|---|---|---|
| | SM?KC | SM?KR | SM?KL | SM?KF | SM?KX |
| NONE<br>See NOTE 1<br>Slot is empty; cannot issue tokens using Credit (6.4), Management (6.5), or Manufacturing (6.6) commands. | PENDING | NO CHANGE (error) | NO CHANGE (error) | NO CHANGE (error) | NO CHANGE (error) |
| PENDING<br>Can issue tokens using VKs loaded in previous key management session. | PENDING | LOADING | NO CHANGE (error)<br>Or PENDING; see NOTE 4 | ACTIVE<br>(or NONE<br>if no keys) | NONE<br>See NOTE 3 |
| LOADING<br>VKs are deleted on entering this state. Vending firmware cannot issue tokens, but manufacturing firmware can use manufacturing (6.6) commands. | PENDING (See NOTE 2) | NO CHANGE (error) | LOADING | ACTIVE,<br>(or NONE<br>if no keys)<br>See NOTE 5 | NONE |
| ACTIVE<br>Can issue tokens using VKs loaded in most recent key management session. | PENDING | NO CHANGE (error) | NO CHANGE (error) | NO CHANGE (error) | NONE |

NOTE 1 All KEK Slots will be in state NONE when the HSM is received by the customer.

NOTE 2 In vending firmware (only) SM?KC in LOADING state is equivalent to SM?KF followed by SM?KC. This behaviour activates the slot so that VKs loaded during the current session can be used.

NOTE 3 Use SM?KX when all VKs need to be deleted. If trying to recover from an aborted key management session (an SM?KC was issued, but have not received a KLF) the following approach is recommended:

Issue another SM?KC to start a new key management session; or

In vending firmware use SM?KF to back out of the key management session and restore the slot to the ACTIVE state.

NOTE 4 In manufacturing firmware (only) SM?KL can be issued in the PENDING state; the HSM will use the KEK loaded by the last successful SM?KR command, and will remain in the PENDING state.

NOTE 5 Avoid issuing SM?KF in manufacturing firmware; it is unnecessary for normal operation.

### 4.5.5 VendingKey (VK) registers

VKs are used to generate STS tokens as defined in IEC 62055-41.

During a key loading session each VK is loaded into the HSM and stored in a VK register. The VK register is automatically allocated when the VK is loaded, and is identified by a register number. The register stores the key material, meta-data such as key attributes (see Annex F for details) and counters for risk management.

The HSM implements a storage hierarchy so that VKs loaded from a KMC are associated with the KEK slot for that KMC, and are isolated from the VKs associated with other KMCs.

A VK is uniquely identified in the STS ecosystem by the combination of its SupplyGroupCode (SGC) and KeyRevisionNumber (KRN). An attempt to store two VKs having the same SGC+KRN combination will overwrite the currently stored VK, even if they originate from different KMCs

### 4.5.6 Key refresh

The first KLF received from a KMC will contain the VKs that are authorized to the HSM at that point in time. In time authorization to use additional VKs may be given, or the risk management rules may cause the existing VKs to expire, or reach vending limits. When new VKs are available, or existing VKs are nearing their limits, it will be required to perform a refresh by completing another key management session (see 4.5.1) with the KMC(s).

As before, the session is initiated by the SM?KC command which puts the KEK slot into the PENDING state (see 4.5.4). In PENDING state it is possible to continue to issue tokens using the VKs already in the HSM, until the new KLF has been received from the KMC and issued the SM?KR command. If the HSM has manufacturing firmware then PENDING state will allow the use of SM?KL to load VKs from the old/existing KLF.

### 4.5.7 ROM key slots

Manufacturing firmware supports operations using a DecoderInitializationTransferKey (DITK), also known as a “dispenser ROM key”. These keys are stored in ROM key slots, which are separate from KEK slots and VK registers. There is one ROM key slot per EncryptionAlgorithm (EA) supported by the HSM. See 6.6.2 for further information on VK attributes that limit key use.

The HSM recognises and enforces business rules that are conveyed in the VK attributes (see Annex F). The following rules are supported:

- ACT and IUT attributes limit the time period for which the VK can be used. Once the IUT is reached, the VK can no longer be used for any purpose until it is refreshed for a new time period (see 4.5.6). The IUT thus provides passive key revocation that controls the maximum duration of unauthorised vending.

- ULM and CLM attributes limit the total value of STS credit tokens (Class=0) that can be issued using the VK. Once the limits are reached the VK can no longer be used to issue credit tokens until it is refreshed with new limits (see 4.5.6). These attributes thus control the maximum financial risk of unauthorised vending.

- CLU attribute associates (clusters) the VK with all other VKs having the same CLU. The HSM will not generate a key change token from a VK in one cluster, to a VK in another cluster. This allows VKs with different owners to coexist in the HSM with no possibility of moving meters between their supply groups. (see Annex C for details).

- SBM attribute limits the STS credit token (Class=0) sub-classes that can be issued using the VK. If a supply group is used for only one resource type (for example electricity) this attribute can prevent the HSM from issuing credit tokens for other resource types.

More technical details for each attribute are given in Annex F.

VK attributes are managed at the KMC and communicated to the HSM in a KLF. The supply group owner must specify the values of these attributes when completing the relevant KMC forms to create a VK or to authorise use of a VK.

Together these attributes can effectively manage the risk of unauthorised vending, but these attributes are optional and no protection is provided if the attributes are not set.

The onus is on the SupplyGroup owner to set the attributes at the KMC.

#### 4.5.8 TokenID-to-RTC window

The TokenID-to-RTC window is a 7 day range around the HSM's real-time clock (RTC). Attempts to issue an STS token with a TokenID outside the window will be rejected; this ensures that the date and time of the token is "close to" the HSM's RTC. This feature can prevent accidental (or deliberate) vending of old or wildly wrong tokens.

The window does not apply to SpecialReservedTokenIdentifier.

The window is fixed at a 7 day period and cannot be disabled.

## 5 Communication protocol

### 5.1 Communication interfaces

See the communication reference model given in 4.4 and Figure 1

The client application process sends a service request message to the HSM application process, which replies with a service response message as given in clause 0

### 5.2 Request message

A request message comprises the concatenation of the 4 fields given in Table 4 with no delimiters between the fields.

**Table 4: Request message fields**

| Field | Representation | Comment |
|---|---|---|
| Header | 5-ASCII character RequestID | Command reference as given in clause 0 |
| Payload | PTVD encoded (variable length) | As given in 5.4 |
| Checksum | 4 hexadecimal digits | See below |
| Terminator | 1 ASCII character | <CR> |
| NOTE 1 In some messages Payload is not present<br>NOTE 2 In some messages Payload is not PTVD encoded. These are so indicated in the relevant commands given in clause 0 | | |

Payload varies in accordance with each command as given in clause 0.

Checksum is calculated over the Header and Payload fields using the standard CRC16/ARC algorithm (see the Rocksoft Model). CRC16/ARC is calculated using the polynomial 0x8005 = X16 + X15 + X2 + 1 and initial value 0x0000. The input to the CRC is a sequence of 8-bit bytes,

and bits are read from each byte Least Significant Bit first. The CRC output is 2 bytes, Least Significant Byte first. The CRC is then BASE16 encoded to produce Checksum. As an example: using the standard test input string “123456789” yields Checksum = “BB3D”.

NOTE Further information on CRC16/ARC may be found at http://reveng.sourceforge.net/crc-catalogue/16.htm#crc.cat-bits.16 .

Checksum yields a 16-bit hexadecimal value (i.e. 4 hexadecimal digits). Each hexadecimal digit is converted into an ASCII character (“0” – “F”) representing the 4 hexadecimal digits.

The Terminator character is always <CR>.

### 5.3 Response message

#### 5.3.1 General

A response message comprises the concatenation of the 5 fields given in Table 5 with no delimiters between the fields.

**Table 5: Response message fields**

| Field | Representation | Comment |
|---|---|---|
| Header | 5-ASCII character ResponseID | Modified (see below) command reference as given in clause 0 |
| Status | 2 ASCII digits | Indicates success or error condition (See below) |
| Payload | PTVD encoded (variable length) | As given in 5.4 |
| Checksum | 4 hexadecimal digits | See below |
| Terminator | 1 ASCII character | <CR> |
| NOTE 1 In some messages Payload is not present<br>NOTE 2 In some messages Payload is not PTVD encoded. These are so indicated in the relevant commands given in clause 0 | | |

The response Header is the same as the request Header except that the '?' is replaced with a '!'. For example, if the request Header was SM?DI, the response Header will be SM!DI. The only exception to this rule is covered in 5.3.3, where some error conditions set the response Header to “GL!ER”.

Status shall be set to 00 for a success condition and to any value between 01 and 99 for an error condition as given in Annex E. For certain commands having extended error conditions, and where so indicated, Status may be set to “EE” (see also 5.3.4).

Payload varies in accordance with each command as given in clause 0.

Checksum is calculated over the Header and Payload fields using the standard CRC16/ARC algorithm (see the Rocksoft Model). CRC16/ARC is calculated using the polynomial 0x8005 = X16 + X15 + X2 + 1 and initial value 0x0000. The input to the CRC is a sequence of 8-bit bytes, and bits are read from each byte Least Significant Bit first. The CRC output is 2 bytes, Least Significant Byte first. The CRC is then BASE16 encoded to produce Checksum. As an example: using the standard test input string “123456789” yields Checksum = “BB3D”.

NOTE Further information on CRC16/ARC may be found at http://reveng.sourceforge.net/crc-catalogue/16.htm#crc.cat-bits.16 .

Checksum yields a 16-bit hexadecimal value (i.e. 4 hexadecimal digits). Each hexadecimal digit is converted into an ASCII character (“0”– “F”) representing the 4 hexadecimal digits.

Terminator character is always <CR>.

#### 5.3.2 Successful response with no error

The successful response is as given in 5.3.1 for a success condition. Some examples are given below, where the blue indicates the value of Header, the grey indicates the value of Status, the green indicates the value of Payload and the yellow indicates the values of Checksum and Terminator.

SM?DI9A51<CR>
SM!DI00P94001234~PSTS65V00~749F<CR>

### 5.3.3 Error response with standard error

When an error occurs during processing of the request, an error code in the range “01” to “99” is returned in the Status field. The Payload field must be omitted with such a standard error code. The possible error codes are described in Annex E. An example of a request with a standard error response is given below.

SM?GAN12~8602<CR>
SM!GA22CA00<CR>

Error code = 22 : CSP_RECORD_EMPTY (VK reg number12 does not exist).

In certain cases, an error can be returned where the response Header is set to “GL!ER” regardless of what the request Header was. This is generally used for errors where the request format/integrity checking fails. A few examples are given below.

SM?DI0000<CR>
GL!ER20A624<CR> Error status = 20 : CHECKSUM_ERROR

SM?JUNK420B<CR>
GL!ER2166E5<CR> Error status = 21 : INVALID_REQUEST_HEADER

### 5.3.4 Error response with extended error

A response with an extended error has the Status field set to “EE” and, unlike standard errors, it includes a Payload field of variable length. Payload contains a single PTVD encoded printable string which is a text description of the error.

The example below shows an extended error in response to a SM?KC request. For simplicity, only the response is shown.

SM!KCEEPSM.1B.1: Load Request speed limit enforced; try again in 60 seconds~3C0A<CR>

## 5.4 PTVD field-encoding of payloads

Unless otherwise noted (applies only to certain commands retained for legacy compatibility) all payloads are encoded using the PrintableTypeValueDelimiter (PTVD) scheme as defined below.

- Separating field types and representations into a serialisation library is a standard practice;
- Typed fields with delimiters improves safety of handling data and protects against type mishandling, buffer errors, etc.
    - Positional rather than tagged arguments, matches the function interface of most languages, and it is straightforward to build/parse messages;
    - Typed fields improve data safety;
    - Delimiter fields, where the delimiter is outside the field’s character set, have equivalent safety to length-prefixed fields, and are easier for humans to construct and to read;
        - Length prefix is required for string fields because the delimiter is within the field’s alphabet.
- Makes implementation easier: a small number of types that are encoded and parsed in a consistent fashion, rather than ad-hoc types;

- More flexible: parameter ranges can be altered without fundamentally changing the field representation.

The PTVD format described in Table 6 is based on SerialTypeLengthValue (STLV) field-encoding and chooses type prefixes that are compatible (non-conflicting) with that encoding.

**Table 6: PTVD field-encoding**

| Type | Detail and encoding | Examples |
|---|---|---|
| Number<br>“N” | Signed 32-bit integer (range -0 – 2147483647) given as a Base-10 number encoded in (ASCII) decimal digits (x’30–x’39).<br>Encoding: “N” 1-10 decimal digits “~”<br>The character “N” followed by the least number of decimal digits [x’30 – x’39] required (minimum 1, maximum 10) to give the value in base-10 (decimal) representation, then the end-of-field character “~”.<br>Leading zeroes are not permitted (the value must be encoded in the least number of digits possible, minimum 1).<br>Negative numbers are not supported. | 0 -> N0~<br>1 -> N1~<br>99 -> N99~<br>2147483647 -> N2147483647~<br><br>N01~ is an invalid encoding (leading zero not permitted). |
| PrintableString<br>“P” | A string as a sequence of 0 to 65535 printable ASCII characters, each character in the range x’20–x’7D.<br>Encoding: “P” chars “~”<br>The character “P” followed by 0 to 65535 characters (x’20–x’7D) giving the value, then the end-of-field character “~”.<br>The empty string is allowed.<br>Note that the end-of-field character “~” is not permitted in the value. | “My KMC Name” -><br>PMy KMC Name~ |
| DateTime “D” | A date and time in the UTC time zone using the Gregorian calendar, given as (ASCII) decimal digits (x’30-x’39).<br>Encoding: “D” CCYYMMDDhhmmss “~”<br>The character “D” followed by 14 decimal digits [x’30–x’39] giving the date parts as detailed below, then the end-of-field character “~”.<br>Each date part must be encoded in exactly the size specified, adding leading zero digits as required:<br>• CCYY: the 4-digit year (with century), range 1900–9999.<br>• MM: the 2-digit month, range 01–12.<br>• DD: the 2-digit day, range 01 to the number of days in the month (max. 31).<br>• hh: the 2-digit hour, range 00–23.<br>• mm: the 2-digit minute, range 00–59.<br>• ss: the 2-digit second, range 00–59.<br>Leap years are supported; leap seconds are not.<br>Note that the value is a valid ISO 8601 “point in time” representation. | 3 Jan 2014, 15h40m54s -><br>D20140103154054~ |
| HexOctets “H” | An octet string as a sequence of 0 to 32767 octets (sometimes called “binary data”), with each octet encoded as a pair of (ASCII) hexadecimal digits (‘0’-‘9’, ‘A’-‘F’).<br>Encoding: “H” hex “~”<br>The character “H” followed by 0 to 65534 hexadecimal digits [x’30–x’39, x’41–x’46] giving the value, then the end-of-field character “~”.<br>Each octet [range 0–255] is represented as two hexadecimal digits giving the octet as a Base-16 value;<br>i.e sequence of octets <219, 45, 82> = octet string x’DB2D52 = base-16 string “DB2D52”<br>Note that the length of the hexadecimal string is twice the length of the octet string. | x’3D7EC8 -> H3D7EC8~ |

# 6 Command reference

## 6.1 General

This clause defines the services of the HSM, including the request and response message structures (excl. Checksum & Terminator parts) and the required behaviour of each service.

## 6.2 General & diagnostic commands

### 6.2.1 SM?DI – Get identification

#### 6.2.1.1 Usage

This command queries the HSM identifier and the firmware identifier. See 4.2 for further definition of these identifiers.

#### 6.2.1.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?DI” | The request header that identifies this service. |

#### 6.2.1.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!DI” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| MID | PrintableString | The module identifier (MID) as given in 4.2 |
| FWID | PrintableString | The firmware identifier (FWID) as given in 4.2 |

#### 6.2.1.4 SM?DI example

SM?DI
SM!DI00P94001234~PSTS65V00~

### 6.2.2 SM?QI – Query identification

#### 6.2.2.1 Usage

This command queries the HSM unique identifier, hardware identifier, firmware identifier and firmware hash.

#### 6.2.2.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?QI” | The request header that identifies this service. |

#### 6.2.2.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!QI” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| IDSM | PrintableString | The HSM’s public key identifier (IDSM) as defined in 4.2, as a “PKID.1” record |
| HWID | PrintableString | The HSM’s hardware identifier as defined in 4.2 (constructed from the 3 elements: MANUFACTURER, MODEL and REVISION). |
| FWID | PrintableString | The HSM’s firmware identifier (FWID) as given in 4.2. |

| | | |
|---|---|---|
| FirmwareHash | HexOctets | A 64-bit hash (or checksum) over the HSM firmware It is meant to uniquely identify the firmware build. This cannot be relied on for any security purpose. |

#### 6.2.2.4 SM?QI example

SM?QI
SM!QI00PSMID.1:Prism:94000933:20170424T094750Z:B6B7687425372936:33D3~PPrism-TSM250-1~PSTS65V01~H06B79CDFEAD42B3F~

### 6.2.3 SM?QD – Query date

#### 6.2.3.1 Usage

This command queries the date and time of the HSM's real time clock (RTC); it also returns the TID window.

The HSM RTC is allowed to drift from the system clock of the host that runs the vending or manufacturing software, but never by more than 7 days (see 4.5.8) The HSM RTC is set at manufacturing time and cannot be changed (other than by sending the HSM to the HSM manufacturer to be reinitialised).

#### 6.2.3.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?QD" | The request header that identifies this service. |

#### 6.2.3.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal "SM!QD" | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| RtcTime | DateTime | Current date and time of the HSM's real time clock. |
| WindowSize | Number | The allowed number of days between the RTC and TokenID. This value is fixed at 7 days and cannot be disabled (See TokenID-to-RTC window in 4.5.8). |

#### 6.2.3.4 SM?QD example

SM?QD
SM!QD00D20040303154054~N7~

### 6.2.4 SM?QL – Query log

#### 6.2.4.1 Usage

This command retrieves a specific entry from the HSM's log file if it exists. By calling this function repeatedly, all log entries can be retrieved.

#### 6.2.4.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?QL" | The request header that identifies this service. |
| Position | Number | The position of the entry in the AuditLog, where 0 is the most recent, 1 is the next most recent, and so on. The maximum is determined by the particular hardware platform. |

#### 6.2.4.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!QL” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| Timestamp | DateTime | The timestamp at which this AuditLog entry was created (taken from the RTC of the HSM). |
| EventType | Number | Indicates the type of event. There is no standard set of event types and is left to the HSM manufacturer to implement as he sees fit |
| Message | PrintableString | The human-readable content of the AuditLog entry. |
| NOTE If an entry exists at the indicated position then the return status is 00 followed by the log entry. If an entry does not exist at the indicated position then the return status is LOG_RANGE_ERROR (45) to indicate end-of-log | | |

#### 6.2.4.4 SM?QL example

SM?QLN0~
SM!QL00D20040303153831~N51~PSMPM completed. ALL CSPS ERASED.~

SM?QLN1~
SM!QL00D20040303153831~N51~PTX Counter incr +10000, Bal = 10000~

### 6.2.5 GL?EC – Echo data

#### 6.2.5.1 Usage

This command echoes back data sent to the HSM; it can be used to test communications.

This command does not use PTVD representation. It is intended for basic communications testing only.

#### 6.2.5.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “GL?EC” | The request header that identifies this service. |
| Delay | 2 ASCII digits | Delay value in seconds before sending the response message. If the delay value is “00” then the response message is returned immediately. (Range “00”-“99”) |
| EchoDataLen | 3 ASCII digits | Number of characters to echo. Must be in the range 000-512 |
| EchoData | ASCII characters | Data to echo, containing only printable string characters (ANS, range x’20-x’7E); length in characters is given by the EchoDataLen field (maximum 512). |

#### 6.2.5.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “GL!EC” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| EchoDataLen | 3 ASCII digits | Number of characters to echo. Must be in the range 000-512 |
| EchoData | ASCII characters | Echo data containing printable string characters (ANS, range x’20-x’7E); length in characters is given by the EchoDataLen field (maximum 512). |

#### 6.2.5.4 GL?EC example

GL?EC00005**12345**
GL!EC00005**12345**

GL?EC00057**12345abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ**
GL!EC00057**12345abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ**

#### 6.2.6 Removed

Legacy command “SM?ID” (from STS600-8-1) removed from this API. HSMs may support this command but it limits the length and representation of the MID and FWID values compared to what is allowed by STS600-4-2.

### 6.2.7 GL?RS – Reset device

#### 6.2.7.1 Usage

This command resets the HSM: all current activities are halted and the device is reset to a state in which it is ready to receive and process commands.

NOTE This command is not intended for use during normal operation. Only use this command to simulate physically unplugging and reconnecting the USB connection to the HSM.

#### 6.2.7.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “GL?RS” | The request header that identifies this service. |

#### 6.2.7.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “GL!RS” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |

#### 6.2.7.4 GL?RS example

GL?RS
indeterminate response - depends on behaviour of the particular HSM under reset conditions

## 6.3 Operational key management commands

### 6.3.1 SM?GL – List KEK slots

#### 6.3.1.1 Usage

This command fetches a list of KEK slot identifiers. It is typically used in conjunction with SM?KQ (see 6.3.2) which fetches information about a single KEK slot, given a slot identifier. For more information see also 4.5.3.

#### 6.3.1.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?GL” | The request header that identifies this service. |

#### 6.3.1.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!GL” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |

| | | |
|---|---|---|
| KekSlotId | A sequence of PrintableString | The KEK slot identifier (see 4.5.3); up to 40 characters. This field will be omitted if there are no non-empty KEK slots; Each KekSlotId value will be unique, but the identifiers are not returned in any particular order.<br>Each element of the sequence is an instance of KekSlotID |
| NOTE Typical use of this command is to obtain a list of KEK slots and then to issue an SM?KQ command for fetching information about each slot (see 6.3.2). | | |

SM?GL examples:

SM?GL
SM!GL00PKMCnameUpTo40chars~ (1 KMC)

SM?GL
SM!GL00PKMCname01~Pkmc_name02~Pkmc_name03~ (3 KMCs)

SM?GL
SM!GL00 (no KMCs)

### 6.3.2 SM?KQ – Query KEK slot

#### 6.3.2.1 Usage

This command fetches information about a KEK slot. To fetch information about all KEK slots, iterate over the list of slot identifiers returned by SM?GL (see 6.3.1).

The information available for a slot includes the identity of the KMC, and the list of VK registers associated with this KEK. For more information see 4.5.3 and 4.5.5.

This command can be used in conjunction with SM?GA to get a complete picture of the keys in the HSM.

#### 6.3.2.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?KQ” | The request header that identifies this service. |
| KekSlotId | PrintableString | The KEK slot identifier (see 4.5.3); up to 40 characters. |

#### 6.3.2.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!KQ” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| State | PrintableString | One of the identifiers PENDING, LOADING, or ACTIVE; indicating the state of this slot (see 4.5.4). |
| KmcId | PrintableString | The name/identifier of the KMC associated with this slot. See also 4.5.3 for more information regarding the possible values of KmcId. When the KEK is established (using SM?KC) the KmcId for the slot is set as the MID part of the Subject field (type PKID) of the KMC’s PUBKEY; the KmcId is thus an IDENT (up to 40 chars; see 5.1.3 of STS 600-4-2). |
| $TVP_{KMC}$ | PrintableString | Current $TVP_{KMC}$: The nonce used to establish the KEK in this slot, under which VKs can be loaded using SM?KL (See 7.3 of STS 600-4-2). This field has a TIMESTAMP (see 5.1.4 of STS600-4-2) value when a KEK is present (in the LOADING state; and sometimes the PENDING state in manufacturing firmware), and is empty at all other times. |

| | | |
|---|---|---|
| Fingerprint | HexOctets | Current Fingerprint: Fingerprint of the KMC's $PUBKEY_{KMC}$ most recently used to establish a KEK in this slot. Fingerprint is set by a successful SM?KR command, and remains until a subsequent successful SM?KR (even after the KEK is destroyed by SM?KF). 16 hex characters; may be empty (as in the transition from NONE to PENDING state). |
| $NewTVP_{KMC}$ | PrintableString | New $TVP_{KMC}$: in the PENDING state this is the nonce used to issue the VKLOADREQ, formatted as an STS600-4-2 TIMESTAMP value. Empty in all other states. |
| NewFingerprint | HexOctets | New Fingerprint: in the PENDING state this is the Fingerprint value of the KMC's $PUBKEY_{KMC}$ used to issue the VKLOADREQ, as 16 hex characters. Empty in all other states. |
| VkReg | A sequence of Number | The identification number of a VK register associated with this KEK slot. This field will be omitted if the slot has no associated VKs;. The register numbers are not ordered, but will be unique.<br>Each element of the sequence is an instance of VkReg |
| NOTE 1 Use SM?GL (see 6.3.1) to get a list of KEK slot identifiers.<br>NOTE 2 Typical usage of this command is to obtain a list of VK register numbers and to then issue an SM?GA command for each VK register.<br>NOTE 3 If a particular KEK slot cannot be located KEK_SLOT_NOT_FOUND_ERROR (27) will be returned.<br>NOTE 4 NewFingerprint and $NewTVP_{KMC}$ have values in the PENDING state only, and reflect the contents of the most recently issued VKLOADRESP for this slot/KMC. Once the corresponding VKLOADRESP is received and successfully loaded using SM?KR, these values are copied to Fingerprint and $TVP_{KMC}$, and then erased.<br>NOTE 5 The $TVP_{KMC}$ value can be used to identify the KLF (, see 4.5.1), from which VK records can be loaded into the HSM (using SM?KL). When $TVP_{KMC}$ is empty there is no KEK in the slot, and VKs cannot be loaded. | | |

#### 6.3.2.4 SM?KQ examples

The examples below show a KEK slot in various states. Notice how the $TVP_{KMC}$, Fingerprint, $NewTVP_{KMC}$ and NewFingerprint fields may be valid or blank depending on the state. If any VKs have been loaded, the VK register numbers will be listed at the end of the response.

SM?KQPKMCname1~
SM!KQ00PPENDING~PKMCname1~P~H~P20130401T120011Z~H8E22231BC992010A~

SM?KQPKMCname1~
SM!KQ00PPENDING~PKMCname1~P~H8E22231BC992010A~P20130401T120011Z~H8E22231BC992010A~

The above 2 examples are both for the PENDING state. The first example has the (current) Fingerprint field blank, indicating that this is the first time that this KMC is being introduced to this HSM. The second example has the (current) FingerPrint field populated, indicating that this KMC has previously been used on this HSM.

The (current) TvpKmc field will always be blank for vending firmware in the PENDING state. However, for manufacturing firmware, the (current) TvpKmc field may be populated indicating that a current KEK is present and can be used for loading.

SM?KQPKMCname1~
SM!KQ00PLOADING~PKMCname1~P20160331T093427Z~H9E1C3307032C6A0F~P~H~N1~N3~N4~N5~N9~N10~N11~N13~

NOTE Per the above LOADING state example, the NewTVPKMC and NewFingerprint fields are blank and the (current) TVPKMC and Fingerprint fields are valid. This will always be the case in the LOADING state.

SM?KQPKMCname1~
SM!KQ00PACTIVE~PKMCname1~P~H9E1C3307032C6A0F~P~H~N1~N2~N3~N4~N5~N6~N7~N8~N9~N10~N11~N12~N13~N14~N15~N16~N17~N18~N19~N20~N21~N22~

In the ACTIVE state as shown above, the (current) $TVP_{KMC}$ field will always be blank, indicating that the KEK has been deleted.

### 6.3.3 SM?KC – Generate VK load request

#### 6.3.3.1 Usage

Generates and returns a VKLOADREQ (see 7.3 of STS 600-4-2). The HSM accepts a KMC public key, and then follows the steps specified in 4.5.1 to establish a KEK which is stored in an available KEK slot (along with meta-data about the key negotiation in 4.5.3).

The VKLOADREQ should be sent to the KMC, which will respond with a KLF containing VKs that can be used by the HSM. See 4.5.1 for details of the key management process.

NOTE This command may return extended error information via status code 'EE' (see 5.3).

#### 6.3.3.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?KC" | The request header that identifies this service. |
| KmcPubKey | PrintableString | The $PUBKEY_{KMC}$ of the KMC (see 4.5.1)..This value is obtained from the KMC. |

#### 6.3.3.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal "SM!KC" | The response header that identifies this service. |
| Status | 2 ASCII digits or "EE" | Result of processing (service status) – 2 ASCII digits interpreted as a decimal: 00 = success, 01-99 = error (see Annex E); or "EE" (Extended Error; see 5.3.4) |
| KekSlotId | PrintableString | The identifier of the KEK slot allocated to (or reused for) this key management session (see 4.5.3); up to 40 characters. |
| Vkloadreq | PrintableString | The VKLOADREQSM (see 7.3 of STS 600-4-2) that should be sent to the KMC. |

NOTE 1 Speed limit (security enhancement, see clause 11 of STS 600-4-2): returns an error if an SM?KC for the same KMC (i.e. same KekSlotId) has been issued within the last 60 seconds.

NOTE 2 If the firmware process (as specified in clause 11 of STS600-4-2) fails then the relevant failure code as specified by clauses 11 and 13 of STS600-4-2 is returned as an Extended Error (Status = "EE").

NOTE 3 The KEK slot is identified by the MID part of the Subject field of the KMC's $PUBKEY_{KMC}$. If a slot with this identity exists it is reused, otherwise a new slot is allocated. If no slots are available, error KEK_SLOT_NOT_FOUND_ERROR (27) is returned. An existing slot can be in any state (PENDING, LOADING, or ACTIVE).

NOTE 4 On success, updates the KEK slot state to PENDING (see 4.5.4).

NOTE 5 Software developers should be aware that the KMC's $PUBKEY_{KMC}$ will change periodically as the STS600-4-2 standard limits the lifetime of a PUBKEY to 3 years.

#### 6.3.3.4 SM?KC example

SM?KCPPK.ECDH.1|KMCID.1:SoftwareV2.3:KMCName:20130301T173830Z:8E22231BC992010A:A4A4|046B540E54A9FE9315B7456852D261B5A42364C4F5AF9693DB0D695A03E6E06C696B2AB03C4613334A53FDD5D543701FAA28156423157500281057 4B7748B327C0329B0F5C1205158B229DB6F0AE5A065CDB64E62BC131A1C434DEFEF46A567BD7|99991231T115959Z|||5AA6~

SM!KC00PKMCName~PVKLOAD.REQ.1|SMID.1:Prism:12345678:20130303T153830Z:829DB2CB4141BBBA:8552|KMCID.1:SoftwareV2.3:KMCName:20130301T173830Z:8E22231BC992010A:A4A4|20130303T153831Z|Prism-TSM250-1|STS65V00|0404DFC9D6464028EB3D718CD8BA3246F3B32E2CD040C8DAA2CEC46ADD57823E8828B33103FE46EC7C7700A3CB22CECDF7B9211E15EF826F22C38D48B6C93A875BB617AB2696F3C27217FA20D98070CA0004C8D8A67F66A463FFD1B50415EC2206|4D8D3283875E86CCEF4328FF5B520B370FA281CDAD19DA6A|67DF~

Request fields are:

– PUBKEY rectype (PK.ECDH.1)
– PKID (rectype, manuf(SWID), UID(KMCID), Serial, Fingerprint, CRC)
– Q_kmc (public key)
– Expiry (followed by 2 empty fields for Issuer and Signature)
– CRC

### 6.3.4 SM?KR – Load VK load response

#### 6.3.4.1 Usage

Loads in a VKLOADRESP (see 7.4 of STS 600-4-2). The VKLOADRESP is sent from the KMC in a KLF. The HSM accepts the VKLOADRESP, and then follows the steps specified in 4.5.1 to finish establishing the KEK (which puts the slot into the LOADING state). After the KEK is established, the VKs (also in the KLF) can be loaded using SM?KL (see 6.3.5).

See 4.5.1 for details of the key management process.

NOTE This command also deletes the existing VKs associated with this KEK slot in the HSM.

This command may return extended error information via status code 'EE' (see5.3).

#### 6.3.4.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?KR" | The request header that identifies this service. |
| Vkloadresp | PrintableString | The $VKLOADRSP_{KMC}$ record received from the KMC. |

#### 6.3.4.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal "SM!KR" | The response header that identifies this service. |
| Status | 2 ASCII digits or "EE" | Result of processing (service status) – 2 ASCII digits interpreted as a decimal: 00 = success, 01-99 = error (see Annex E);or "EE" (Extended Error; see 5.3.4) |
| KekSlotId | PrintableString | The identifier of the KEK slot used for this key management session (see 4.5.3); up to 40 characters. |

NOTE 1 Prerequisite: has to have generated a VKLOADREQ using SM?KC, to which this VKLOADRESP is a response.

NOTE 2 The information in the VKLOADRESP is used to locate the KEK slot which was allocated for this key management session by the SM?KC command. If the KEK slot cannot be located a KEK_SLOT_NOT_FOUND_ERROR (27) will be returned. The KEK slot must be in the PENDING state.

NOTE 3 If the firmware process (as specified in clause 11 of STS600-4-2) fails then the relevant failure code as specified in clauses 11 and 13 of STS600-4-2 is returned as an extended error (Status = "EE").

NOTE 4 On success, updates the KEK slot state to LOADING (see 4.5.4).

#### 6.3.4.4 SM?KR example

SM?KRPVKLOAD.RESP.1|KMCID.1:ArbSWID:ArbKMCID:20040101T120000Z:65AF90C4DBAF23D3:3AC4|SMID.1:Prism:12345678:20130303T153830Z:E7C42215AF1FE951:9C7D|20130303T153839Z|A9FE9315B7456852D261B5A42364C4F5AF9693DB0D6931B6|9D28~SM!KR00PArbKMCID~

Request fields are:

– Rectype ("VKLOAD.RESP.1")
– PKID_KMC
– PKID_SM
– TVP

– MacTag_kmc
– CRC

### 6.3.5 SM?KL – Load VK

#### 6.3.5.1 Usage

Loads a VK encrypted under a KEK using the HSM process defined in clause 11 of STS600-4-2. The VK is contained in a WRAPPED-KEY record sent from the KMC in a KLF. Before using this command the KEK must be loaded from the same KLF (in the VKLOADRESP record) using SM?KR (see 6.3.4).

If the HSM has vending firmware then load the required VKs from the KLF, and then close the key management session using SM?KF (see 6.3.6).It is not possible to use the VKs to generate STS tokens until the session has been closed, but once the session is closed no more keys (protected under the KEK from this KLF) can be loaded.

If the HSM has manufacturing firmware it is not necessary to close the key management session, the STS manufacturing commands (see 6.6) can be used while the session is open (KEK slot is in LOADING state).

See 4.5.1 for details of the key management process.

#### 6.3.5.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?KL” | The request header that identifies this service. |
| KekSlotId | PrintableString | The identifier of the KEK slot used for this key management session (see 4.5.3); up to 40 characters. |
| WrappedKey | PrintableString | The VK, as a WRAPPED-KEY record (see 7.5 of STS600-4-2) from the KLF. |

#### 6.3.5.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!KL” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| KeyRegister | Number | The number of the key register allocated to the VK. |
| Attributes | PrintableString | The attributes of the VK, in card format (see 7.5.1 of STS 600-4-2). Contents of this field are as for SM?GA (see 6.3.8). |

NOTE 1 Prerequisite: a KEK must have been established using SM?KC , and SM?KR commands. The KEK slot must be in the LOADING state.

NOTE 2 With manufacturing firmware (only) the KEK slot may be in the PENDING or LOADING state. In the PENDING state this command will use the KEK from the last successful SM?KR command.

NOTE 3 Use KekSlotId returned by the SM?KR command. If the KEK slot cannot be located a KEK_SLOT_NOT_FOUND_ERROR (27) will be returned.

NOTE 4 The VK is stored in a register (see 4.5.5). A VK register is allocated automatically. If no registers are available, the command will fail with STS_VK_REG_NOT_FOUND_ERROR (28). Use SM?KD (see 6.3.9) to free up a register.

NOTE 5 A VK is uniquely identified in the STS ecosystem by the combination of its SGC and KRN. An attempt to store two VKs having the same SGC+KRN combination will overwrite the currently stored VK, even if they originate from different KMCs (see 4.5.5).

NOTE 6 The HSM supports the VK attributes listed in Annex F. Unrecognised attributes are ignored. Most of the attributes are stored in the VK register for later use (refer to Annex F or to SM?GA command).

NOTE 7 The KEK slot state is not affected by this command (see also 4.5.4).

#### 6.3.5.4 SM?KL example

SM?KLPKMC1~PKEY.1|00000000000000000000000A|ACT20130716T151031Z;BDT20130716T151031Z;DKG04;KEN255;KRN6;KTC2;SGC0000758825;|42A08D17E89E5B9DE6F863E13D044853B49880FE8560C449B61A77986079DB14E015734E|73C7~

SM!KL00N1~PACT20130716T151031Z;BDT20130716T151031Z;CLMFFFFFFFF;CLU0;DKG04;IUT20380119T031407Z;KCV92E101;KEN255;KRN6;KSLKMC1;KTC2;SBMFFFF;SGC0000758825;ULM999999999;VND0;~

Request fields (wrapped key):

– Rectype (“KEY.1”)
– Nonce
– Attributes
– Encrypted key
– crc

### 6.3.6 SM?KF – VK loading finished

#### 6.3.6.1 Usage

Finishes the key management session (see 4.5.1). The KEK that was established for this session is deleted as required by clause 13 of STS600-4-2.

If the HSM has vending firmware it is not possible to use the VKs to generate STS tokens until the session has been closed, but once the session is closed no more keys (protected under the KEK from this KLF) can be loaded.

If the HSM has manufacturing firmware it is not necessary to close the key management session: the STS manufacturing commands (see 6.6) can be used while the session is open (KEK slot is in LOADING state).

This command can also be used to abort a key management session that has been started by SM?KC command, but for which no VKLOADRESP has been received (so the slot is in the PENDING state). Issuing an SM?KF on the KEK slot returns the slot to the ACTIVE state without affecting existing VKs (whereas an SM?KX command will delete existing VKs).

NOTE Once the session is closed it is not possible to load more VKs under this KEK; a new key management session would have to be started. Load all the VKs needed before closing the session.

#### 6.3.6.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?KF” | The request header that identifies this service. |
| KekSlotId | PrintableString | The identifier of the KEK slot used for this key management session (see 4.5.3); up to 40 characters. |

#### 6.3.6.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!KF” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| NOTE 1 Prerequisite: a VKLOADREQ must have been generated using SM?KC , and optionally established the KEK using SM?KR. The KEK slot must be in the PENDING or LOADING state.<br>NOTE 2 Use KekSlotId returned by the SM?KC or SM?KR command. If the KEK slot cannot be located a KEK_SLOT_NOT_FOUND_ERROR (27) will be returned.<br>NOTE 3 On success, updates the KEK slot state to ACTIVE if there are VKs associated with this slot, else to NONE (see 4.5.4). This command will not cause VK registers to be cleared. | | |

#### 6.3.6.4 SM?KF example

SM?KFPKMC1~
SM!KF00

### 6.3.7 SM?KX – Delete KEK slot and associated VKs

#### 6.3.7.1 Usage

Deletes the identified KEK slot and clears all associated VK registers.

#### 6.3.7.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?KX” | The request header that identifies this service. |
| KekSlotId | PrintableString | The identifier of the KEK slot used for this key management session (see4.5.3); up to 40 characters. |

#### 6.3.7.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!KX” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| NOTE 1 Use SM?GL (see 6.3.1) to get a list of KEK slot identifiers.<br>NOTE 2 If the KEK slot cannot be located a KEK_SLOT_NOT_FOUND_ERROR (27) will be returned.<br>NOTE 3 Use SM?KD (see 6.3.9) to clear a single VK register.<br>NOTE 4 On success, the KEK slot is deleted and its effective state is NONE (see 4.5.4).<br>NOTE 5 See also 4.5.1 for details of the key management process. | | |

#### 6.3.7.4 SM?KX example

SM?KXPKMC1~
SM!KX00

### 6.3.8 SM?GA – Get VK attributes

#### 6.3.8.1 Usage

This command fetches a VK’s attributes (see 4.5.5).

#### 6.3.8.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?GA” | The request header that identifies this service. |
| KeyRegister | Number | The number of the VK register to query (the first register is 1). The range is firmware platform specific. |

#### 6.3.8.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!GA” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |

| | | |
|---|---|---|
| Attributes | PrintableString | All attributes that the HSM knows for the key, in card format (see 7.5.1 of STS 600-4-2). See Annex F for a list of attributes. All required attributes will be present; optional attributes may be present or absent. |
| NOTE 1 Use SM?KQ (see 6.3.2) to get a list of VK register numbers associated with a KEK slot.<br>NOTE 2 Using an out-of-range key register will result in error PTVD_RANGE (63).<br>NOTE 3 Attempting to retrieve the attributes from an empty VK register will result in error CSP_RECORD_EMPTY (22).<br>NOTE 4 The HSM firmware supports all attributes defined in 7.5.1 of STS600-4-2, plus additional attributes detailed in Annex F. | | |

#### 6.3.8.4 SM?GA example

SM?GAN13~
SM!GA00PACT20040301T140000Z;BDT19930101T000000Z;CLMFFFFFFFF;CLU0;DKG04;IUT20150301T140000Z;KCV6C3DE4;KEN255;KRN1;KSLTestKMCName;KTC2;SBMFFFF;SGC0000123457;ULM999999999;VND1;~

### 6.3.9 SM?KD – Delete VK

#### 6.3.9.1 Usage

This command clears the indicated VK register (see 4.5.5).

#### 6.3.9.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?KD” | The request header that identifies this service. |
| KeyRegister | Number | The number of the VK register to clear (the first register is 1). The range is firmware platform specific; |

#### 6.3.9.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!KD” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| NOTE 1 If an attempt is made to clear an empty key register the response status will be success (0); it is thus safe to delete empty registers.<br>NOTE 2 If the last VK register associated with a KEK slot that is in the ACTIVE state is cleared, then the slot is deleted implicitly (as if by SM?KX , see 6.3.7). | | |

#### 6.3.9.4 SM?KD example

SM?KDN13~
SM!KD00

## 6.4 STS transfer credit commands

### 6.4.1 General

Credit transfer commands are only available in vending HSMs with vending firmware.

### 6.4.2 SM?VC – Vend STS credit token

#### 6.4.2.1 Usage

Constructs an STS Class 0 (credit) token and encrypts the token under a DecoderKey (that is derived for a specific meter from the VK).

#### 6.4.2.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?VC” | The request header that identifies this service. |
| KeyRegister | Number | The number of the VK register to use (the first register number is 1). The range is platform firmware specific; |
| MeterPan | PrintableString | The 18 digit meter PrimaryAccountNumber that uniquely identifies the meter, as specified in 6.1.2 of IEC 62055-41. MeterPAN must include valid DRNCheckDigit and PANCheckDigit parts. |
| TI | Number | The TariffIndex on which the meter is configured, as given in 6.1.7 of IEC 62055-41; range 0 to 99. |
| EA | Number | The EncryptionAlgorithm supported by the meter, as given in 6.1.5 of IEC 62055-41. See Annex A for supported algorithms. |
| TCT | Number | The TokenCarrierType supported by the meter and used for this transaction, as given in 6.1.3 of IEC 62055-41. See also Annex B. |
| SubClass | Number | The required token SubClass, which indicates the resource and units in which TransferAmount is denominated. The values supported are as given in 6.2 of IEC 62055-41 |
| Amount | HexOctets | This is the encoded Amount content, not the decimal number of units.<br>For SubClass 0 – 3 the encoding is given in 6.3.6.2 of IEC 62055-41, resulting in a 16-bit (2 octet) input (range x’0000 to x’FFFF<br>For SubClass 4 – 7 the encoding is given in 6.3.6.3 of IEC62055-41, resulting in a 20-bit (3 octet) input (range x’000000 to x’FFFFFF). |
| TID | HexOctets | TokenIdentifier as given in 6.3.5 of IEC 62055-41. The TID ensures that each token is unique and prevents the meter from accepting a token more than once. The TID must be computed relative to the VK’s BaseDate. Range is x’00 to x’FFFFFF. |

#### 6.4.2.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!VC” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see see Annex E) |
| TokenHex | PrintableString | Encrypted Token as 17 hex characters. |
| TokenDec | PrintableString | Encrypted Token as 20 decimal digits. |

NOTE 1 Vending may be restricted by the VK attributes that limit key use (see 6.6.2).

NOTE 2 The legacy HSMs required a trailing space on the MeterPAN instance, which is no longer permitted;

NOTE 3 The HSM does not generate tokens for reserved SubClass 8 – 15.

NOTE 4 The TID should reflect the vending system time as the number of minutes elapsed since the VK’s BaseDate (BDT attribute in Appendix B of STS600-4-2); i.e. TID is relative to BDT. If the TID differs from the HSM’s RTC by more than the TID-to-RTC window of 7 days (see 4.5.8) then the transaction will fail.

NOTE 5 If the TID is a SpecialReservedTokenIdentifier instance, then the TID-to-RTC window has no effect.

NOTE 6 TransferAmount and TID fields use a HexOctets format rather than Number so that they can be easily understood as exactly which bits are packed into the token.

NOTE 7 When the VK is a VCDK (see 6.5.2.2.4 of IEC 62055-41) the TCT must be “01” (STS magnetic token carrier) or else the operation will fail.

#### 6.4.2.4 SM?VC examples

The following example uses these inputs:

KeyRegister = 1 ; MeterPAN = 600727000000000009; TI = 1; EA = 7; TCT = 1; SubClass = 0; TransferAmount = 0x0001 ; TID = 0x599868 (1Mar2004,14h00).

SM?VCN1~P600727000000000009~N1~N7~N1~N0~H01~H599868~
SM!VC00P1ED633E9B406C83BA~P35552328719914533818~

The next example is for a currency token; use the inputs above with the following changes:

SubClass = 4; Amount = 1000.00000 currency base units = 0x011FF4 encoded TransferAmount (see 6.3.6.3 of IEC62055-41).

SM?VCN1~P600727000000000009~N1~N7~N1~N4~H011FF4~H599868~
SM!VC00P1A9422C2DC1D23D4F~P30643103389619207503~

NOTE STS tokens contain random data and are thus not predictable; it should not be expected to see the same token values when these examples are executed.

## 6.5 STS management commands

### 6.5.1 General

The STS management commands produce meter-specific management tokens.

### 6.5.2 SM?VM – Vend STS management function token

#### 6.5.2.1 Usage

Constructs an STS Class 2 (meter-specific management) token and encrypts the token under a DecoderKey (that is derived for a specific meter from the VK).

#### 6.5.2.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?VM” | The request header that identifies this service. |
| KeyRegister | Number | The number of the VK register to use (the first register is 1). The range is platform firmware specific; |
| MeterPan | PrintableString | The 18 digit meter PrimaryAccountNumber that uniquely identifies the meter, as specified in 6.1.2 of IEC 62055-41. MeterPAN must include valid DRNCheckDigit and PANCheckDigit parts. |
| TI | Number | The TariffIndex on which the meter is configured, as given in 6.1.7 of IEC 62055-41; range 0 to 99. |
| EA | Number | The EncryptionAlgorithm supported by the meter, as given in 6.1.5 of IEC 62055-41. See Annex A for supported algorithms. |
| TCT | Number | The TokenCarrierType supported by the meter and used for this transaction, as given in 6.1.3 of IEC 62055-41. See also Annex B. |
| SubClass | Number | The required token SubClass, which indicates the management function to be executed. The values supported are given in Table 7 (see also 6.2 of IEC 62055-41 and clause 4 of STS 202-5) |
| TransferValue | HexOctets | This is a 16-bit value corresponding to the relevant token field for the particular SubClass instance as given in Table 7<br>Range x’0000 to x’FFFF |
| TID | HexOctets | The TokenIdentifier as given in 6.3.5 of IEC 62055-41. The TID ensures that each token is unique and prevents the meter from accepting a token more than once. The TID must be computed relative to the VK’s BaseDate. Range is x’00 to x’FFFFFF. |

**Table 7: SubClass instances supported for SM?VM command**

| SubClass | Management function | TransferValue |
|---|---|---|
| 0 | SetMaximumPowerLimit | MPL field in 6.2.4 of IEC 62055-41 |
| 1 | ClearCredit | Register field in 6.2.5 of IEC 62055-41 |
| 2 | SetTariffRate | Rate field in 6.2.6 of IEC 62055-41 |
| 5 | ClearTamperCondition | Pad field in 6.2.9 of IEC 62055-41 |
| 6 | SetMaximumPhasePowerUnbalanceLimit | MPPUL field in 6.2.10 of IEC 62055-41 |

| | | |
|---|---|---|
| 7 | SetWaterMeterFactor | WMFactor field in 6.2.11 of IEC 62055-41 |
| 9 | *Reserved for future use* | *Unspecified* |
| 10 | Extended token set | Index, FlagIndex and FlagValue fields in 4.2 of STS 202-5<br>Or<br>Index and ControlValue fields in 4.3 of STS 202-5 |
| 11 – 15 | *Proprietary use* | |

#### 6.5.2.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!VM” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| TokenHex | PrintableString | Encrypted Token as 17 hex characters. |
| TokenDec | PrintableString | Encrypted Token as 20 decimal digits. |
| NOTE 1 Vending may be restricted by the VK attributes that limit key use (see 6.6.2).<br>NOTE 2 The HSM does generate tokens for proprietary SubClass 11 – 15.<br>NOTE 3 See also SM?VC notes on MeterPAN, TID and TransferAmount; they are all applicable to this command.<br>NOTE 4 When the VK is a VCDK (see 6.3.5 in IEC 62055-41) the TCT must be 01 (STS magnetic token carrier) or else the operation will fail. | | |

#### 6.5.2.4 SM?VM example

The following example uses these inputs: KeyRegister = 1 ; MeterPAN = 600727000000000009; TI = 1; EA = 7; TCT = 1; SubClass = 0; TransferAmount = 0x0001; TID = 0x599868 (1Mar2004,14h00).

SM?VMN1~P600727000000000009~N1~N7~N1~N0~H01~H599868~
SM!VM00P21C2F281872BC752C~P38924374189855765804~

NOTE STS tokens contain random data and are thus not predictable; it should not be expected to see the same token values when this example is executed.

### 6.5.3 SM?VK – Vend STS key change token

#### 6.5.3.1 Usage

Constructs a set of key change tokens; each token is encrypted under a DecoderKey (that is derived for a specific meter from the VK). Key change tokens are a special sub-set of STS Class 2 (meter-specific management) tokens that can change the meter’s DecoderKey and configuration (SGC, KRN, RO, KEN and TI).

The HSM enforces several rules that are necessary to protect the meter’s revenue stream (that is to prevent key change tokens from being used to bypass the meter’s credit controls). These rules are described in Annex C.

#### 6.5.3.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?VK” | The request header that identifies this service. |
| KeyRegisterOld | Number | The number of the VK register that contains the meter’s current VK (the first register is 1). The range is platform firmware specific; |

| | | |
|---|---|---|
| KeyRegisterNew | Number | The number of the VK register that contains the VK to which the meter should be moved (the first register is 1). The range is platform firmware specific; |
| MeterPan | PrintableString | The 18 digit meter PrimaryAccountNumber that uniquely identifies the meter, as specified in 6.1.2 of IEC 62055-41. MeterPAN must include valid DRNCheckDigit and PANCheckDigit parts. |
| TIOld | Number | The TariffIndex on which the meter is currently configured, as given in 6.1.7 of IEC 62055-41; range 0 to 99. |
| EA | Number | The EncryptionAlgorithm supported by the meter, as given in 6.1.5 of IEC 62055-41. See Annex A for supported algorithms. |
| TCT | Number | The TokenCarrierType supported by the meter and used for this transaction, as given in 6.1.3 of IEC 62055-41. See also Annex B. |
| TINew | Number | The TariffIndex to which the meter should be changed, as given in 6.1.7 of IEC 62055-41; range 0 to 99 . |
| NumTokens | Number | Indicates the expected number of key change tokens (KCTs) in the result set. When EA=11 there are always 4 KCTs; when EA=7 there are usually 2 KCTs, but some meters accept an optional 3rd KCT. These are the only permitted values. |

#### 6.5.3.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!VK” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| Rollover | Number | Indicates the value of the RolloverKeyChange bit in the SubClass=2 token (that is always one of the output tokens as given in 6.3.20 of IEC 62055-41). |
| NumTokens | Number | Indicates the number of key change tokens actually generated; will always equal the NumTokens value from the request. |
| TokensHex | PrintableString | Encrypted tokens as (NumTokens x 17) hex characters; each token is exactly 17 hex characters. |
| TokensDec | PrintableString | Encrypted tokens as (NumTokens x 20) decimal digits; each token is exactly 20 decimal digits. |
| NOTE 1 The VK attributes that limit key use (see 6.6.2) apply to both the current (KeyRegisterOld) and new (KeyRegisterNew) keys.<br>NOTE 2 This operation is subject to the key change rules given in Annex C.<br>NOTE 3 See also SM?VC notes on MeterPAN; they are applicable to this command.<br>NOTE 4 When the VK is a VCDK (see 6.5.2.2.4 in IEC 62055-41) the TCT must be 01 (STS magnetic token carrier) or else the operation will failSM?VK examples | | |

The following example requests **2** tokens for an EA07 meter (VKs may use DKGA=02 or DKGA=04).

SM?VKN1~N1~P600727000000000009~N1~N7~N2~N2~N**2**~SM!VK00N0~N**2**~P2CCA0882E5
72FDBAD0D530672AB6C65C22~P5163842306004273450915361891762113502242~

The following example requests **3** tokens for an EA07 meter (VKs may use DKGA=02 or DKGA=04).

SM?VKN1~N1~P600727000000000009~N1~N7~N2~N1~N**3**~SM!VK00N0~N**3**~P0B69A53CA9
0241C38172746EA7D561873A12E44131AB241A95F~P131579213907831102002669408255
845030482621780554703251286367~

The next example requests **4** tokens for an EA11 meter (VKs must both use DKGA=04).

SM?VKN21~N22~P600727000000000009~N1~N11~N2~N2~N4~SM!VK00N0~N4~P262F59E
E012E530560CB4A259770F8F7C72CCD5FEAFF3282BC73D0E8618DD13FB7D9~P4402426
84176570245981464856207034161351151653471532847999943703936212373434713~

### 6.5.4 SM?VT – Verify encrypted STS token

#### 6.5.4.1 Usage

Decrypts an encrypted STS token (using a DecoderKey that is derived for a specific meter from the VK), verifies the token's cyclic redundancy code (CRC) as given in 6.3.7 of IEC 62055-41, and returns information about the token.

#### 6.5.4.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?VT" | The request header that identifies this service. |
| KeyRegister | Number | The number of the VK register to use (the first register is 1). The range is platform firmware specific |
| MeterPan | PrintableString | The 18 digit meter PrimaryAccountNumber that uniquely identifies the meter, as specified in 6.1.2 of IEC 62055-41. MeterPAN must include valid DRNCheckDigit and PANCheckDigit parts. |
| TI | Number | The TariffIndex on which the meter is configured, as given in 6.1.7 of IEC 62055-41; range 0 to 99. |
| EA | Number | The EncryptionAlgorithm supported by the meter, as given in 6.1.5 of IEC 62055-41. See Appendix A for supported algorithms. |
| TokenDec | PrintableString | Encrypted Token as 20 decimal digits. |

#### 6.5.4.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal "SM!VT" | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E). Possible error values include: KEY_ISSUE_EXPIRED_ERROR, KEY_NOT_YET_ACTIVE, VENDING_INHIBITED, KMC_KEY_STATE_ERROR. |
| ValidationResult | Number | Indicates whether or not the token is valid.<br>0 = valid<br>Non-zero value (invalid) is an error code. Possible invalid values are TOKEN_CRC_ERROR, KEY_EXPIRED_ERROR, DDTK_CREDIT_ERROR.<br>If invalid then all subsequent fields will be set to zeros. |
| Class | Number | The token's Class, as given in 6.3.2 of IEC 62055-41; range 0 – 3. Credit tokens have Class=0; management tokens have Class=2. |
| Subclass | Number | The token's SubClass, as given in 6.3.3 of IEC 62055-41 and STS 202-5; range 0 – 15. |
| TransferValue | HexOctets | This is a 16-bit value corresponding to the relevant token field for the particular Class and SubClass instances as given in Table 8<br>Range x'0000 to x'0FFFFF |
| TID | HexOctets | The token's TID value, which indicates the time of issue relative to the VK's BaseDate; range x'000000 to x'FFFFFF. |
| NOTE 1 If the token is not valid then the Class, SubClass, TransferValue, and TID fields will contain zeros.<br>NOTE 2 If the token's Class & SubClass fields indicate that it is a key change token then the TransferValue and TID fields will have a value equal to zero.<br>NOTE 3 It is not possible to verify manufacturing key change tokens generated by SM?MK, (see 6.6.6) using this command.<br>NOTE 4 For a proprietary management token (Class = 2 and SubClass = 11 to 15) as given in Table 8, TransferValue is returned as zero.<br>NOTE 5 See also SM?VC for notes on MeterPAN, TID and TransferValue; they are all applicable to this command. | | |

**Table 8: Class and SubClass instances supported for SM?VT command**

| Class | SubClass | Function | TransferValue |
|---|---|---|---|
| 0 | 0 | Electricity units credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 1 | Water units credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 2 | Gas units credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 3 | Time units credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 4 | Electricity currency credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 5 | Water currency credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 6 | Gas currency credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 7 | Time currency credit | Amount field in 6.2.2 of IEC 62055-41 |
| 0 | 8 – 15 | *Reserved* | |
| 2 | 0 | SetMaximumPowerLimit | MPL field in 6.2.4 of IEC 62055-41 |
| 2 | 1 | ClearCredit | Register field in 6.2.5 of IEC 62055-41 |
| 2 | 2 | SetTariffRate | Rate field in 6.2.6 of IEC 62055-41 |
| 2 | 3 | Set1stSectionDecoderKey | 6.2.7.2 of IEC 62055-41 |
| 2 | 4 | Set2ndSectionDecoderKey | 6.2.7.3 of IEC 62055-41 |
| 2 | 5 | ClearTamperCondition | Pad field in 6.2.9 of IEC 62055-41 |
| 2 | 6 | SetMaximumPhasePowerUnbalanceLimit | MPPUL field in 6.2.10 of IEC 62055-41 |
| 2 | 7 | SetWaterMeterFactor | WMFactor field in 6.2.11 of IEC 62055-41 |
| 2 | 8 | Set3rdSectionDecoderKey | 6.2.7.4 of IEC 62055-41 |
| 2 | 9 | Set4thSectionDecoderKey | 6.2.8.5 of IEC 62055-41 |
| 2 | 10 | Extended token set | Index, FlagIndex and FlagValue fields in 4.2 of STS 202-5<br>Or<br>Index and ControlValue fields in 4.3 of STS 202-5 |
| 2 | 11 – 15 | *Proprietary use* | *Specified by meter manufacturer* |

#### 6.5.4.4 SM?VT example

SM?VTN1~P600727000000000009~N1~N7~P35552328719914533818~
SM!VT00N0~N0~N0~H000001~H599868~

### 6.6 STS manufacturing commands

#### 6.6.1 General

Manufacturing commands are only available in manufacturing firmware. These commands provide the functionality required by meter manufacturers to work with DITKs, also known as “dispenser ROM keys”.

This functionality requires the HSM to have a dedicated hardware port for key component entry separate from the command interface port. Manufacturing firmware is not permitted on HSMs that do not have a dedicated hardware port for key component entry.

STS manufacturing commands are not available in the vending firmware.

#### 6.6.2 DITK management (informative)

IEC 62055-41 provides for a DITK that is owned and managed by the meter manufacturer. The DITK is used to initialise the meter’s DecoderKey register during production or repair at the manufacturer’s premises. Before leaving the manufacturer’s premises, the DITK must be

replaced with a unique (DUTK), common (DCTK) or default (DDTK) key that is derived from a VK.

The DITK must be handled securely, with respect to the principles of dual control and split knowledge. To this end the established practice from retail financial standards have been implemented:

- Each DITK is split into components, such that it is possible to recover the key given all the components, but not possible to recover any knowledge of the key given one or more (but not all) components.
- Each component is entrusted to a different custodian. A custodian is a trusted individual who keeps the component safe and secret.
- The integrity of the key is verified by means of a check value.

The manufacturing firmware provides the following features to manage and use DITKs:

- Storage for the DITKs in the form of ROM key slots. There is one slot for each EA supported by the HSM (see Annex A for supported EAs; Use SM?MG (see 6.6.3) to generate the key components for a DITK. The size of the DITK is determined by the EA with which it will be used. The following will be needed:
    - A key component entry device (KCED) connected to the HSM's CSP port;
    - Two or three custodians who will own these key components. Each custodian must record the EA, the key's check value, and the key component allocated to them.
- Use SM?ML (see 6.6.4) to load the DITK into the ROM key slot in the HSM. The following will be needed:
    - A KCED connected to the HSM's CSP port;
    - The key's custodians with their key components.
- It is possible to query each ROM key slot using SM?MQ (see 6.6.5);
- To change a meter's DecoderKey from the DITK to a DUTK or DCTK use SM?MK (see 6.6.6).

### 6.6.3 SM?MG – Generate and display key components

#### 6.6.3.1 Usage

Generates the DITK components and displays them on the KCED connected to the HSM's CSP port (see 6.6.2 for information on when and how to use this command).

A KCED must be connected to the HSM's CSP port prior to issuing this command.

#### 6.6.3.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal "SM?MG" | The request header that identifies this service. |
| EA | Number | The EncryptionAlgorithm supported by the meters that will use this DITK (as given in 6.1.5 of IEC 62055-41). This field determines the length of the key & components, and the algorithm used to compute KeyCheckValue.<br>See also Annex A for supported algorithms and key lengths. |
| NumComps | Number | The number of key components to generate; must be 2 or 3.<br>Each component must be kept by a different custodian, so this parameter reflects or determines the number of custodians. |

#### 6.6.3.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!MG” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| KeyCheckValue | HexOctets | The check value of the generated key (3 octets / 6 hex characters). |
| NOTE 1 A KCED must be connected to the HSM’s CSP port before issuing this command. Once the command has been issued, follow the instructions on the KCED.<br>NOTE 2 For the STA EncryptionAlgorithm (EA07) the DITK will be generated with odd parity. | | |

The key components are generated such that combining all components using XOR will yield the DITK.

#### 6.6.3.4 SM?MG example

SM?MGN7~N2~
SM!MG00HD5D44F~

### 6.6.4 SM?ML – Load dispenser ROM key

#### 6.6.4.1 Usage

Load a dispenser ROM key (DITK) into a ROM key slot. The DITK is entered in the form of components using a KCED connected to the HSM (see 6.6.2).

The length of each key component is determined by the EncryptionAlgorithm (EA); see Annex A for supported encryption algorithm details.

A KCED must be connected to the HSM prior to issuing this command.

#### 6.6.4.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?ML” | The request header that identifies this service. |
| EA | Number | The EncryptionAlgorithm supported by the meters that will use this DITK (as given in 6.1.5 of IEC 62055-41); identifies the ROM key slot and determines the length of the key (and components) and the algorithm used to compute/verify KeyCheckValue.<br>See Annex A for supported algorithms and key lengths. |
| NumComps | Number | The number of key components for this DITK; must be 2 or 3.<br>Each component must be kept by a different custodian, so this parameter also reflects the number of custodians. |
| KeyCheckValue | HexOctets | The check value of the DITK (3 octets / 6 hex characters). |

#### 6.6.4.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!ML” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| KeyCheckValue | HexOctets | The check value of the DITK (3 octets / 6 hex characters). |

NOTE 1 It is possible to use SM?MG (see 6.6.3) to generate and display appropriate key components prior to using this command.

NOTE 2 A KCED must be connected to the HSM before issuing this command. Once the command has been issued follow the instructions on the KCED.

NOTE 3 The components are combined using XOR function to form the DITK.

NOTE 5 Updates the audit log to indicate that a DITK has been loaded (logs EA and KeyCheckValue).

#### 6.6.4.4 SM?ML example

SM?MLN7~N2~HD5D44F~
SM!ML00HD5D44F~

### 6.6.5 SM?MQ – Query dispenser ROM key slots

#### 6.6.5.1 Usage

This command fetches information about a ROM key slot (see also 6.6.2).

#### 6.6.5.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?MQ” | The request header that identifies this service. |
| EA | Number | The EncryptionAlgorithm supported by the meters that will use this dispenser ROM key (as given in 6.1.5 of IEC 62055-41); identifies the ROM key slot and determines the algorithm used to compute KeyCheckValue.<br>See Annex A for supported algorithms. |

#### 6.6.5.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!MQ” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| KeyCheckValue | HexOctets | The check value of the DITK (3 octets / 6 hex characters). |
| NOTE Returns error STS_CSP_RECORD_EMPTY (22) if no DITK is loaded for the specified EA. | | |

#### 6.6.5.4 SM?MQ example

SM?MQN7~
SM!MQ00HD5D44F~

### 6.6.6 SM?MK – Generate manufacturer key change token

#### 6.6.6.1 Usage

Constructs a set of key change tokens using a dispenser ROM key; the tokens will change a DecoderKey from the DITK to a DDTK, DUTK, or DCTK as given in 6.5.2.4 of IEC 62055-41.

#### 6.6.6.2 Request format

| Field | Representation | Description |
|---|---|---|
| RequestID | Literal “SM?MK” | The request header that identifies this service. |
| KeyRegisterNew | Number | The number of the VK register that contains the VK to which the meter should be moved (the first register is 1). The range is platform firmware specific; |

| | | |
|---|---|---|
| MeterPan | PrintableString | The 18-digit meter PrimaryAccountNumber that uniquely identifies the meter, as specified in 6.1.2 of IEC 62055-41. MeterPAN must include valid DRNCheckDigit and PANCheckDigit parts. |
| TINew | Number | The TariffIndex to which the meter should be changed, as given in 6.1.7 of IEC 62055-41; range 0 to 99. |
| EA | Number | The EncryptionAlgorithm supported by the meter, as given in 6.1.5 of IEC 62055-41. Also identifies the dispenser ROM key slot to use. See Annex A for supported algorithms. |
| NumTokens | Number | Indicates the expected number of key change tokens (KCTs) in the result set. When EA=11 there are always 4 KCTs; when EA=7 there are usually 2 KCTs, but some meters accept an optional 3rd KCT. These are the only permitted values. |

#### 6.6.6.3 Response format

| Field | Representation | Description |
|---|---|---|
| ResponseID | Literal “SM!MK” | The response header that identifies this service. |
| Status | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| NumTokens | Number | Indicates the number of key change tokens actually generated; will always equal the NumTokens value from the request. |
| TokensHex | PrintableString | Encrypted tokens as (NumTokens x 17) hex characters; each token is exactly 17 hex characters. |
| TokensDec | PrintableString | Encrypted tokens as (NumTokens x 20) decimal digits; each token is exactly 20 decimal digits. |

NOTE 1 The VK attributes that limit key use (see 6.6.2) apply to the new (KeyRegisterNew) VK.

NOTE 2 The key change rules in Annex C do not apply to this command. The HSM however ensures that the destination VK (in KeyRegisterNew) is not expired (by checking the IUT attribute).

NOTE 3 SM?MK never sets the RO bit in the key change token, but it will work with a VK of any BaseDate value.

NOTE 4 The legacy HSMs required a trailing space after MeterPAN; this is no longer permitted or required.

NOTE 5 SM?VT cannot verify tokens created by this command.

#### 6.6.6.4 SM?MK examples

The following example requests **2** tokens for an EA07 meter (VK may use DKGA=02 or DKGA=04):

SM?MKN1~P600727000000000009~N1~N7~N**2**~SM!MK00N**2**~P0423D5B02B3EB6312287C340B1118CF858~P0477307124724920398646676222092245792856~

The following example requests **3** tokens for an EA07 meter (VK may use DKGA=02 or DKGA=04):

SM?MKN1~P600727000000000009~N1~N7~N**3**~SM!MK00N**3**~P1AF3F7E26152111B7287C340B1118CF85822B7C9DAA9072E19E~P3107469465590805752746676222092245792856400270409440555749 42~

The following example requests **4** tokens for an EA11 meter (VK must use DKGA=04):

SM?MKN20~P600727000000000009~N1~N11~N**4**~SM!MK00N**4**~P3294EBE9D1582232432EE8B1AD908526BD0D762C7A456009FF415F8E00E9F68F2C84~P5831675820674712451658720379100482315965155201867745035960202533218595886957683 6~

STS tokens contain random data and are not predictable; it should not be expected to see the same token values if these examples are executed.

### 6.7 Vending authorisation commands (informative)

See Annex G for more information on these commands.

Annex A
(normative)

# Supported EncryptionAlgorithm values (EAs)

## A.1 Algorithms

The HSM supports the EAs given in Table A.1 and in 6.5.4 and 6.5.6 of IEC 62055-41.

**Table A.1 – Supported EncryptionAlgorithm values (EAs)**

| Value | Description | Key length | KCV algorithm |
|---|---|---|---|
| 7 | EncryptionAlgorithm07 (STA) | 64 bits (16 hex chars) | DES KCV |
| 11 | EncryptionAlgorithm11 (Misty1) | 128 bits (32 hex chars) | IBM SHA256 KCV |
| NOTE The dispenser ROM key components have the same length as the key. The KCV for a DITK is calculated in the same way as for a VK (see below). | | | |

## A.2 KeyCheckValue algorithms

Check Values are meant to assist in identifying a key, contributing to confidence that the expected key material is present. This identification is probabilistic (not unique), and does not contribute to the security of Key Management.

### A.2.1 DES KCV for 64-bit VKs and DITK (DKGA02)

Encrypt a zero block (8 bytes all = x'00) with the key in DES ECB mode; take the leftmost 3 bytes of the result as the check value.

DES keys are comprised of 56 bits of key material and 8 bits of parity (stored as the least significant bit (LSB) of each octet). The DES KCV is insensitive to changes in the parity bits. STA keys have 64 bits of key material, but are treated as DES keys in this algorithm, so this Check Value will not detect if LSBs in an STA key have been modified (for example by parity adjustment).

DES KCV example:

Key = x'0123456789ABCDEF (which has odd parity)
Check value = x'D5D44F

### A.2.2 IBM SHA256 KCV for 128-bit VKs and DITK (DKGA04)

Concatenate the byte x'01 with the key, in that order; compute the SHA-256 hash of this concatenation; take the leftmost 3 bytes of the result as the check value.

IBM SHA256 KCV example:

Key = x'9BCE36CC873C91022268DA69A7115558
Check value = x'895C63

Annex B
(normative)

# Supported TokenCarrierType values (TCTs)

## B.1 TokenCarrierType values

IEC 62055-41 states that TCT is a 2-digit number used to uniquely identify the type of token carrier onto which the token should be encoded for transferring to the payment meter. The HSM returns the token or tokens in decimal and hexadecimal formats. It is the responsibility of the vending system to use the appropriate format and/or encoding as required by the TCT so that the token can be correctly transferred to the meter.

The HSM only uses the TCT to enforce rules of use applicable to a DCTK: “A DCTK shall only be used with payment meters that use erasable magnetic card token carriers (TCT value = 01) and shall only be accepted by such payment meters. Payment meters with any other token carrier types (TCT value > 01) shall reject tokens encrypted under DCTK values.” (see 6.5.2.3.5 of IEC 62055-41 “DCTK: DecoderCommonTransferKey”). This restriction applies to the following commands:

- SM?VC – Vend STS credit token;
- SM?VM – Vend STS management function token;
- SM?VK – Vend STS key change token;
- The restriction *does not apply* to SM?MK – Generate manufacturer key change token as the DITK is allowed to be a parent of a DCTK (see 6.5.2.3.2 of IEC 62055-41 ) and the manufacturer may have a proprietary method to load the key change token into the meter.

Annex C
(normative)

# Key change rules

## C.1 Key change rules

The SM?VK command (see 6.5.3) enforces the following rules when generating a key change token from a source VK (VDDK, VCDK, or VUDK) to a destination VK (VDDK, VCDK, or VUDK).

These rules are applied in order, stopping when the first ALLOW or REJECT is encountered:

1. If the source VK is expired (now > IUT) then REJECT the change;
2. If the destination VK is expired (now > IUT) then REJECT the change;
3. If the source VK and the destination VK are from different KMCs then REJECT the change;
4. If the source VK's cluster (CLU) is not zero and is different to the destination VK's cluster (CLU) then REJECT the change;

NOTE If the source and destination VKs have the same SGC then they are assumed to be in the same cluster; this implies that the source and destination SGCs must be different in order to REJECT the change at this point.

5. If the source VK BaseDate is greater than the destination VK BaseDate then REJECT the change;
6. If the source VK BaseDate equals the destination VK BaseDate then ALLOW the change;
7. Else the source VK BaseDate is prior to the destination VK BaseDate, so ALLOW the change and set the RO bit.

Annex D
(informative)

# HSM hardware platforms

## D.1 Hardware platforms

Some features and command parameter ranges differ between HSM hardware platforms. HSM hardware platform behaviour also differs when the HSM is in a tamper state.

The HSM may also support additional commands, including proprietary commands necessary for the production and maintenance of the HSM, other than those commands provided in this standard.

Please refer to the manuals supplied with the HSM for further information on any additional commands and operating methods.

Annex E
(normative)

# Response message status codes

## E.1 Status codes

The status codes are given in Table E.1.

**Table E.1 – Response message status codes**

| Code | Identifier | Description |
|---|---|---|
| 00 | SUCCESSFUL | The command executed successfully. |
| 01 | DEVICE_FAILURE | There was a hardware or general failure. |
| 02 | FORMAT_ERROR | The format of the command data is incorrect. Can occur if the request contains the incorrect number of parameters. In SM?CI command can occur if the signature length is invalid. |
| 03 | TOKEN_CRC_ERROR | The CRC of the token being verified does not match. Applies to SM?VT only. Can also be used in the ValidationResult field in the SM?VT response. |
| 04 | Reserved for future use | |
| 05 | KEY_TYPE_ERROR | The KeyType of the specified key register does not meet all of the requirements. Applicable to SM?VC command: cannot vend to VDDK (KT=1) and can only vend to VCDK (KT=3) if TCT is 1 (magnetic card). |
| 06 | Reserved for future use | |
| 07 | Reserved for future use | |
| 08-09 | Reserved for future use | |
| 10 | KEYPAD_ENTRY_CANCELLED | KCED entry/display aborted. Applicable to SM?ML and SM?MG commands only. (HSM manufacturing firmware only) |
| 11-19 | Reserved for future use | |
| 20 | CHECKSUM_ERROR | The CRC of the command request is incorrect. This code is only used with a GL!ER response. |
| 21 | INVALID_REQUEST_HEADER | The header of the command request is invalid or the command is not supported. This code is only used with a GL!ER response. |
| 22 | Proprietary | |
| 23 | Proprietary | |
| 24 | Proprietary | |
| 25 | Reserved for future use | |
| 26 | KMC_KEY_STATE_ERROR | The KEK slot is not in the required state (i.e. LOADING, ACTIVE or PENDING) for the requested command. Applies to SM?KR, SM?KL, SM?KF and SM?VC. |
| 27 | KEK_SLOT_NOT_FOUND_ERROR | A KEK slot with the requested KMCID could not be found (SM?KX), or there are no more available slots (SM?KC). |

| | | |
|---|---|---|
| 28 | VK_REG_NOT_FOUND_ERROR | During key loading, could not find a VK register with matching attributes that belongs to the specified parent key. |
| 29 | KEY_CHANGE_NOT_ALLOWED | One of the rules for a key change has not been met. Applies to SM?VK only. |
| 30 | Reserved for future use | |
| 31 | INSUFFICIENT_CREDIT_BALANCE | The credit limit associated with the specified VK has been depleted. Refer to ULM and CLM key attributes. Applies to SM?VC only. |
| 32 | Reserved for future use | |
| 33 | INVALID_SECURITY_MODULE_ID | ModuleID does not match that provided in the certificate. Applies to SM?CI command only. |
| 34-39 | Proprietary | |
| 40 | KEY_PARAM_ERROR | A parameter associated with a VK is outside of its required range. For example EA, TI, DKGA, KT, KRN, Class, SubClass. |
| 41 | KEY_EXPIRED_ERROR | TID > KEN, or KMC key expired. Applies to SM?VC, SM?VM and SM?KC only. Can also be used in the ValidationResult field in the SM?VT response. |
| 42 | KEY_ISSUE_EXPIRED_ERROR | Current module time > IUT. Applies to SM?VC, SM?VM, SM?VK. Can also be used in the ValidationResult field of the SM?VT response. |
| 43 | KEY_NOT_YET_ACTIVE | Current module time < ACT. Applies to SM?KC and SM?VK only. |
| 44 | TID_OUT_OF_WINDOW | Current module time is outside of TokenID-to-RTC window. Applies to SM?VC and SM?VM only. |
| 45 | LOG_RANGE_ERROR | No more entries available to read or log is empty. Applicable to SM?QL. |
| 46-47 | Reserved for future use | |
| 48 | SUBCLASS_NOT_ALLOWED | Specified SubClass not included in the set of allowed SubClass values for this VK. SM?VC only. |
| 49 | DF_FORMAT_ERROR | Error in dfconcat formatting of a request, including an invalid CRC if the dfconcat string has one (see 5.7.1 of STS 600-4-2). Applies to SM?KC, SM?KR and SM?KL commands.<br>Can also occur when formatting a response if the stored data is corrupt or invalid, although this is not expected behaviour. |
| 50 | Proprietary | |
| 51 | Reserved for future use | |
| 52 | KEYPAD_ENTRY_ERROR | An error occurred during KCED entry/display. Applicable to SM?ML and SM?MG commands in HSM manufacturing firmware only. |
| 53 | KEYPAD_NOT_FOUND | A KCED is not connected to the CSP port. Applicable to SM?ML and SM?MG commands in HSM manufacturing firmware only. |
| 54-55 | Proprietary | |
| 56 | Reserved for future use | |
| 57 | TAMPERED_STATE | A tamper event has occurred. This code is only used with a GL!ER response. SM?QL will provide insight. Module must be returned to manufacturer. |
| 58 | APP_ERROR_STATE | Internal error. Power cycling may resolve, otherwise permanent. This code is only used with a GL!ER response. |

| | | |
|---|---|---|
| 59 | Reserved for future use | |
| 60 | PTVD_NFIELDS | Error in PTVD formatted command request. Too many or too few fields, or a partial field was encountered. |
| 61 | PTVD_FIELDTYPE | Error in PTVD formatted command request. A different type to what was expected (for a field in that position of the request). |
| 62 | PTVD_ENCODING | Error in PTVD formatted command request. The encoding of a field is invalid for that field's type. |
| 63 | PTVD_RANGE | Error in PTVD formatted command request. Field decoded ok but the value is outside of the permitted range. |
| 64-66 | Proprietary | |
| 67-75 | Reserved for future use | |
| 76 | INVALID_PAN_ERROR | PrimaryAccountNumber (PAN) is not 18 digits or has an incorrect check digit. Applicable to SM?MK and SM?VK/VC/VM/VT. |
| 77 | AUTHENTICATION_ERROR | Failed to verify expected mac (applies to SM?KL). |
| 78 | CHECK_DIGIT_ERROR | The key check digits (KCV) for the key components entered via the KCED do not match the expected value. Indicates that one or more components have been entered incorrectly. Applicable to the SM?ML command in HSM manufacturing firmware only. |
| 79 | Proprietary | |
| 80 | DDTK_CREDIT_ERROR | Used in the "ValidationResult" field in the SM?VT response if using a default key with a Class 0 token. |
| 81 | Proprietary | |
| 82-84 | Reserved for future use | |
| 85 | INVALID_NUM_TOKENS | The number of tokens requested in the SM?VK command is not valid for the chosen EncryptionAlgorithm (EA). |
| 86 | Proprietary | |
| 87 | INVALID_INSTRUCTION | Signed instruction has invalid parameters. Applicable to SM?CI. |
| 88-93 | Proprietary | |
| 94 | ATTRIBUTE_ERROR | Applies to SM?KL only. A mandatory key attribute (in card format) has not been provided, or an attribute does not meet the parameter constraints for this attribute. |
| 95 | Proprietary | |
| 96 | Proprietary | |
| 97 | Proprietary | |
| 98 | Proprietary | |
| 99 | Proprietary | |
| "EE" | STS_EXTENDED_ERROR | Extended error; see 5.3<br>For the SM?KC and SM?KR commands the extended error message texts correspond to the error messages specified in clauses 11 and 13 of STS600-4-2. |

Annex F
(normative)

# Supported Vk attributes

## F.1 Supported VK attributes

The HSM shall support the VK attributes given in Table F.1.

Some of these attributes are defined in Appendix B of STS 600-4-2, as indicated in the "Description" column, while others are additional that relate to risk management features as defined in Table F.1.

Attributes are used in the following contexts:

- In a WRAPPED-KEY record in a KLF (see 4.5.2); see also 7.5 in STS600-4-2;
- In the response to an SM?GA command (see 6.3.8).

The "Presence" column indicates whether the attribute is permitted or required:

- "Required" indicates that the attribute must be present in all contexts;
- "Optional" indicates that the attribute may be present or absent in any context;
- "Special" indicates that presence differs between contexts; see the attribute's description for details.

Unknown attributes (those not in Table F.1) may appear in a WRAPPED-KEY record in a KLF, but these will be ignored by the HSM. Unknown attributes will never be present in an SM?GA response.

**Table F.1 – Supported VK attributes**

| Name | Content Type | Presence | Description |
|---|---|---|---|
| ACT | TIMESTAMP | Required | ActivationTime: the date and time at which the VK becomes active. (See Appendix B of STS 600-4-2) |
| BDT | TIMESTAMP | Required | BaseDate: the date associated with a TID value of zero. (See Appendix B of STS 600-4-2) |
| CLM | 8H | Optional | CurrencyLimit: credit limit per VK. The content is an IEEE 754 single-precision floating point value in 32-bit binary interchange format encoding (sign bit as the MSB) which is then BASE16 encoded; or "FFFFFFFF" indicating no limit.<br>The VK's credit limit register is set to this value. For a currency credit token the STS TransferAmount is computed (from the Amount) and deducted from the register (the register value must decrease even if the TransferAmount value is below the precision floor of the current exponent).<br>Omit the CLM card to have no limit on vending currency tokens. The HSM returns a value of "FFFFFFFF" for CLM to indicate no limit on the amount of currency that the HSM can vend. Note that the value of CLM returned by the SM?GA command will be the current balance remaining which may be lower than the CLM value that was provided with the SM?KL command.<br>Example: The current amount 102.04000 is represented in IEEE 754 32-bit binary interchange format encoding as the bit string 01000010110011000001010001111011, and the BASE16 encoding of that bit string is "42CC147B".<br>Online calculator: www.h-schmidt.net/FloatConverter/IEEE754.html |

| | | | |
|---|---|---|---|
| CLU | 1-10D | Optional | SGCCluster. When performing a key change, if the current and new keys are not from the same SGC then the key change is only allowed if the current key cluster is 0 or if the current and the new key are on the same cluster (see Annex C). Omitting the CLU card will result in the default value of 0 being assigned to this attribute. Examples: 0, 7, 123, 826401745 |
| DKG | 2D | Required | DecoderKeyGenerationAlgorithm (DKGA). (See Appendix B of STS 600-4-2) |
| IUT | TIMESTAMP | Optional | IssuedUntil: after this date the HSM will prevent the VK from being used for any purpose. (See Appendix B of STS 600-4-2) |
| KCV | 6H | Special | KeyCheckValue for the VK, computed by the HSM. Never present in a KLF; always present in SM?GA and SM?KL response. |
| KEN | 3D | Required | KeyExpiryNumber. (See Appendix B of STS 600-4-2) |
| KRN | 1D | Required | KeyRevisionNumber. (See Appendix B of STS 600-4-2) |
| KSL | IDENT | Special | KeySlotID (can be referred to as KMCName) identifies the KMC from which this VK originates.<br>Never present in a KLF; always present in SM?GA and SM?KL response. |
| KTC | 1D | Required | KeyType. (See Appendix B of STS 600-4-2) |
| SBM | 4H | Optional | SubClass bitmap. Allowed credit token sub-classes, a 16-bit bitmap in the range 0-($2^{16}$-1). If (SBM & 0x0001) is unequal to 0 then SubClass 0 credit tokens are allowed. Omitting the SBM card will result in the default value of FFFF being assigned to this attribute (no SubClass restrictions applied).<br>Examples: 0007 (SubClass 0-3), FFFE (all sub-classes except 0) |
| SGC | 10D | Required | SupplyGroupCode. SGC is a 10 digit decimal value. However, the top 4 digits must be zero because STS has not yet approved more than 6 digits for SGC. The HSM will return an error if the SGC attribute is > 0000999999. (See Appendix B of STS 600-4-2) |
| SGN | 1-99P | Optional | SupplyGroupName. Presence is optional in a KLF; optional in SM?GA or SM?KL response. (See Appendix B of STS 600-4-2) |
| ULM | 1-9D | Optional | UnitLimit: credit limit per VK. A 1 to 9-digit decimal number in the range 0 – 999,999,998. The value of 999,999,999 indicates no limit and will result in the counter never being decremented. The VK's unit limit register is set to the ULM value provided. For a non-currency credit token the STS TransferAmount is computed and deducted from the register.<br>Omit the ULM card to have no limit on vending non-currency tokens (in this case, the HSM returns a value of 999,999,999 for ULM).<br>Note that the value of ULM returned by the SM?GA command will be the current balance remaining which may be lower than the ULM value that was provided with the SM?KL command.<br>Examples: 1000; 398674; 10000000 |
| VND | 1D | Special | VendControl: Set to 1 if vending is allowed, or 0 if vending is not allowed. Vending (SM?VC, SM?VM, SM?VK or SM?VT) will not be possible if any of the following are true: ActivationTime not reached; past IssuedUntil date; past the InhibitVending date set by SM?CI; KEK slot not in a state that allows vending.<br>For vending firmware, the KEK slot must be in the ACTIVE or PENDING state. For manufacturing firmware, the KEK slot can be in any state (ACTIVE, PENDING or LOADING) to allow vending (applies to SM?VM, SM?VK, SM?VT).<br>Never present in a KLF; always present in SM?GA and SM?KL response. |

Annex G
(informative)

# Vending authorization commands

## G.1 Vending authorisation commands

### G.1.1 General

Vending authorisation commands are only applicable to vending firmware.

A Security Module (SM) may optionally implement these commands to manage and limit the number of credit vending transactions the SM may perform and/or the period of time for which the SM may be operated.

The STS Association is not able to certify and takes no responsibility for the certification of the implementation of these commands due to the proprietary nature of the INTX data element contained in the SM?CI command response message.

The implementation and deployment risks associated with these commands need to be addressed within the contractual agreement between the SM supplier and the end user of the SM.

The STS Association does not provide a support service for the operational deployment of a SM that implements these commands. Such support service needs to be agreed between the SM supplier and the end user of the SM.

### G.1.2 SM?CI – Transaction Counter Increment

#### G.1.2.1 Usage

This command loads a Signed Instruction that may be provided by the SM supplier that updates these limits.

#### G.1.2.2 Request format

| Field | Representation | Description |
|---|---|---|
| **Request ID** | Literal “SM?CI” | The request header that identifies this service. |
| **ManufInstr** | HexOctets | The ManufInstr is composed of a 256 octet PKC1 Public Key Certificate, followed by a signed INTX instruction (128 to 208 octets). |

#### G.1.2.3 Response format

| Field | Representation | Description |
|---|---|---|
| **Response ID** | Literal “SM!CI” | The response header that identifies this service. |
| **Status** | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| **TxCounter** | Number | New Transaction Counter balance (transactions remaining) |
| **MID** | Printable string | This is the module identifier (MID) |
| **Nonce** | HexOctets | A unique challenge value (“nonce”, 4 octets) that is required by the SM manufacturer in order to issue a subsequent Signed Instruction. |
| **InhibitVendDate** | DateTime | Date & time after which vending will be inhibited. Use a date of 1 Jan 2038 to indicate that vending should not be time-limited. |

#### G.1.2.4 Availability

| | | | | | |
|---|---|---|---|---|---|
| SM with Vending FW | Yes | SM in Tamper State | No | SM with manufacturing firmware | No |

#### G.1.2.5 Notes:

- The SM supplier can use this command to configure a SM for a maximum TX counter value of 1,999,999,999.
- The supplier can use the Inhibit Vending Date to disable vending after a specified date. This would typically be used on a SM issued for demonstration purposes. To enable vending for an unlimited time period, the date must be set to 1 Jan 2038. The transaction counter (balance) is not affected (or lost) when vending is inhibited. Future SM?CI commands can re-enable vending and increment the transaction counter.

#### G.1.2.6 Example

SM?CIH1089DEBD56F8D098E1D685ED72E3C1946097B9E1CAE911B4FADD367B6B13C65
292DE1F9BE31032EBA477E567A46A92F2C0988B59B55A49EA73A47F0F4D1D12E69B4F2B
47FEE6CA981417F02B69419A5C94C45D30447D8D511B4C8524E269FB5C3AA1695D7FB0F
39C301C41A24EDD556600635D5526DBB8C1481ABC6E8AE1B3967FEFBC2E3D9D68556C0
4649CBE75C3FC5AAECEB2ED73681454772EC434253BBABE750731F33AEF37AB79D2F6F
DA0D52189A79E6AC02FECE03894A9C23846A2B4ABF49112B9E72F71EBA691B3E1AE6C5
2BB7B5AFDD232CA371AAFE9B994591EFF06F273AEA901931C17408D23F00926F645698B
85CCDAEE4D1A29857D254C6C4B2798991E7811CE84B4F6C8049275BE85ED2DE329A0E0
91DB6F532DEB8561CA84F20C625F9860ED3C76216BC0E7D8D02D8841A1B37DC05776AB
4DCB51B933513F7411891FD5EC3CB38CC6A6C38358E518CF3C66D2172D120A17937F9E
27CCE7F7600BFFD507C4B3408B38E9B318EC44251731D01D67E55183E6978E42505E48B
F24C8D876763E4D40BFAB78FC0D13D9AA~

SM!CI00N100000000~P94001234~H56656F83~D20140101220000~

### G.1.3 SM?CQ – Transaction Counter Query

#### G.1.3.1 Usage

This command fetches the limits currently applicable to the SM.

#### G.1.3.2 Request format

| Field | Representation | Description |
|---|---|---|
| **Request ID** | Literal “SM?CQ” | The request header that identifies this service. |

#### G.1.3.3 Response format

| Field | Representation | Description |
|---|---|---|
| **Response ID** | Literal “SM!CQ” | The response header that identifies this service. |
| **Status** | 2 ASCII digits | Result of processing (service status) – 2 ASCII digits interpreted as a decimal. 00 = success, 01-99 = error (see Annex E) |
| **TxCounter** | Number | Current Transaction Counter balance (transactions remaining). Maximum balance is 1,999,999,999. |
| **MID** | Printable string | This is the module identifier (MID) |
| **Nonce** | HexOctets | A unique challenge value (“nonce”, 4 octets) that is required by the SM manufacturer in order to issue a subsequent Signed Instruction. |
| **InhibitVendDate** | DateTime | Date & time after which vending will be inhibited. A date of 1 Jan 2038 indicates that vending is not time-limited. |

#### G.1.3.4 Availability

| | | | | | |
|---|---|---|---|---|---|
| SM with Vending FW | Yes | SM in Tamper State | No | SM with manufacturing firmware | No |

#### G.1.3.5 Notes

A date of 1 Jan 2038 for the Inhibit Vending Date field indicates that vending is not time-limited.

#### G.1.3.6 Example

SM?CQ

SM!CQ00N100000000~P94001234~H56656F83~D20140101220000~