STS Association

STS 600-4-2

Edition 1.3

Nov 2021

STANDARD TRANSFER SPECIFICATION –

Companion Specification – Key Management System

1 Contents

Revision History

Edition	Clause	Date	Change details from previous Edition
1.0			Initial revision
1.1	General	March 2016	See Appendix A.
1.2	Appendix A-K	May 2018	Removed previous Appendix A (change history) and replaced this with a VKLOADRESP example. Removed unused Appendices, renumbered Appendices.
1.3	Appendix I	Feb 2019	Added informative reference to STS600-9-1

STANDARD TRANSFER SPECIFICATION ASSOCIATION

_____

COMPANION SPECIFICATION

STS 600-4-2: Standard transfer specification (STS) – Companion specification –

FOREWORD

The STS Association is a Not for Profit Company registered in terms of South African Law. The organisation holds an annual general meeting of members where the members elect nominated members to the board. The board consists of elected directors as well as one director each from the four founding organisations, Itron, Conlog, Landis+Gyr and Eskom in South Africa. The board is supported by a secretariat provided by the firm VdW&Co in Johannesburg, South Africa.

The Standard Transfer Specification (STS) has become recognized as the only globally accepted open standard for prepayment systems, ensuring inter-operability between system components from different manufacturers of prepayment systems. The application of the technology is licensed through the STS Association, thus ensuring that the appropriate encryption key management practices are applied to protect the security of the prepayment transactions of utilities operating STS systems. It has become established as a de facto worldwide standard for transfer of electricity prepayment tokens since its initial introduction in South Africa in 1993.

It has become established as a worldwide standard for the transfer of electricity prepayment tokens since its introduction in South Africa in 1993 and subsequent publication by the International Electrotechnical Commission as the IEC62055 series of specifications.

Address: The STS Association, P.O. Box 868, Ferndale 2160, Republic of South Africa.
Tel: +27 061 5000
Fax: +27 86 680 7449
Email: email@sts.org.za
Website: http://www.sts.org.za

Introduction

This document specifies a Key Management System (infrastructure) for the Standard Transfer Specification (STS) – as contemplated in [IEC 62055-41] section 9 and Annex A – including all relevant cryptographic techniques, protocols, and data formats.

The infrastructure is intended to:

1. Standardise Security Module (SM) initialisation and vending key transfer from a Key Management Centre (KMC) to an SM.
2. Conform to contemporary standards for key management and cryptographic security, with the expectation that the specified cryptographic techniques may remain in use until the year 2045 ([STS COP 402-1] 2nd roll over Base Date)[1].
3. Enable secure remote coding of SMs to simplify logistical processes.
4. Support the STSA Code of Practice for Token ID rollover [STS COP 402-1].

The security target has been set at 128 bits for the whole system and 192 bits for key management operations, in accordance with the key and algorithm security-strength recommendations of [NIST SP800-57 PART 1] and [NIST SP800-131A].

All cryptographic protocols and algorithms in this specification are standardised by ISO and NIST. Algorithms – other than those prescribed or constrained by [IEC 62055-41] – are approved for US Smart Grid Cyber Security by [NISTIR 7628], and meet or exceed[2] the Augmented Requirements for US Federal Cryptographic Key Management Systems [NIST SP800-152 DRAFT].

This document contains the following information:

- Definitions of the Terms, Abbreviations and Symbols that are used, which should be read in conjunction with the corresponding sections of [IEC 62055-41].
- Key management process diagrams summarising the steps in the initial and operational key management processes.
- Specification of Data Types and Encodings that are used to provide exact representations of logical data fields.
- Definitions of Cryptographic Primitives and Data Formats and Structures used in key management processes.
- Initial key management and trust establishment processes, comprising SM Initialisation and KMC Initialisation.
- The operational key management process, comprising the Vending Key Load Request sent by the SM to the KMC, and the Vending Key Load Response from the

[1] Cryptography is a fast-changing field in which 30+ year predictions carry significant risk. The Key Management techniques should be subject to a security review well before 2035 (3rd Base Date).

[2] This specification has a higher security target than [NIST SP800-152 DRAFT] and thus uses larger key sizes that exceed the security requirement of that standard, but do not meet the interoperability requirements.

KMC (including Vending Key Attributes that may be transferred with the vending key).

2 Normative references

[FIPS PUB 186-3] Digital Signature Standard (DSS), June 2009
HTTP://CSRC.NIST.GOV/PUBLICATIONS/FIPS/FIPS186-3/FIPS_186-3.PDF

[IEC 62055-41] IEC 62055-41:2007 Electricity metering – Payment systems – Part 41: Standard transfer specification (STS) – Application layer protocol for one-way token carrier systems

[ISO 8601] ISO 8601:2004 Data elements and interchange formats – Information interchange – Representation of dates and times

[ISO 10118-3] ISO/IEC 10118-3:2004 Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions

[ISO 11770-3] ISO/IEC 11770-3:2008 Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques

[ISO 14888-3] ISO/IEC 14888-3:2006 Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms

[ISO 15782-1] ISO 15782-1:2009 Certificate management for financial services – Part 1: Public key certificates

[ISO 15946-1] ISO/IEC 15946-1:2008 Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 1: General

[ISO 18033-2] ISO/IEC 18033-2:2006 Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers

[ISO 18033-3] ISO/IEC 18033-3:2010 Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers

[ISO 19772] ISO/IEC 19772:2009 Information technology – Security techniques – Authenticated encryption

[ISO 9797-2] ISO/IEC 9797-2:2011 Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a dedicated hash-function

[NIST SP800-56A] NIST Special Publication 800-56A Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), March 2007
HTTP://CSRC.NIST.GOV/PUBLICATIONS/NISTPUBS/800-56A/SP800-56A_REVISION1_MAR08-2007.PDF

[NIST SP800-108] NIST Special Publication 800-108 Recommendation for Key Derivation Using Pseudorandom Functions, October 2009
HTTP://CSRC.NIST.GOV/PUBLICATIONS/NISTPUBS/800-108/SP800-108.PDF

[RFC 4648] The Base16, Base32, and Base64 Data Encodings, October 2006
HTTP://TOOLS.IETF.ORG/HTML/RFC4648#SECTION-8

[RPT-0032-100] Ziliant Systems, “Review of the updated STS Key Management Specification”, version 1.0, 29 November 2012

3 Definitions, Abbreviations and Symbols

Note: where abbreviations used in this specification are not listed below, they are defined in the cited reference within this specification.

3.1 Definitions

BCD	Packed Binary Coded Decimal [W:BCD]. Each decimal digit is encoded as one nibble (4 bits). For example BCD("1234") = x'1234.
Big Endian	Byte ordering from most significant to least significant. See Wikipedia:Endianness [W:END].
Bit string	A bit string is an ordered sequence of 0's and 1's.
Cryptographic boundary	Continuous perimeter that establishes the physical and/or logical bounds of a Cryptographic Module (Security Module). See [ISO 19790].
Dual Control	From [ISO 15782-1]: Process of utilizing two or more separate entities (usually persons), who are operating in concert, to protect sensitive functions or information
Octet	An eight-bit byte. See Wikipedia:Octet [W:OCT].
Octet string	A variable-length ordered sequence of octets (eight-bit bytes). See [ITU X.680] (ASN.1) and Wikipedia:Octet [W:OCT]. Any bit string with length a multiple of 8 may be interpreted as an octet string (starting from the left of the bit string, each group of 8 bits is an octet).
Split Knowledge	From [ISO 15782-1]: Condition under which two or more entities separately have key fragments which, individually, convey no knowledge of the resultant cryptographic key

3.2 Abbreviations

HSM	Hardware Security Module, also called a Cryptographic Module. This abbreviation usually refers to a Security Module used by the KMC to manage keys and perform cryptographic operations.
IV	Initialisation Vector, used in some block cipher modes of operation. Also called a Starting Variable (SV).
KMC	Key Management Centre, an infrastructure component used to manage keys in an STS system (as in [IEC 62055-41]).
PRF	Pseudorandom function.
RBG	Random Bit Generator such as those defined in [ISO 18031] or [NIST SP800-90].

RTC	Real-time Clock.
SM	Security Module (called a “Cryptographic Module” in [IEC 62055-41]).
LVCONCAT	Length Value Concatenation
HMAC	Hashed Message Authentication Code
MAC	Message Authentication Code

3.3 Symbols

$a \times b$ *or* $a.b$	Integer multiplication; the product of integers a and b.
a / b	Real division; the quotient of *a* divided by *b* as a real number.
$a \div b$	Integer division with truncation; the largest integer *x* where $x \le a/b$.
$[a]$	The ceiling of real number *a*: the smallest integer ≥ *a*. $[a/b] = (a+b-1) \div b$.
$\sum a_i$ for i=1 to n	The sum of values $a_1 + a_2 + \ldots + a_n$.
$a \equiv b \pmod q$	*a* is congruent to *b* modulo *q*.
∅	A null or empty field.
[*n*,*m*]	The interval (range) of integers between and including *n* and *m*.
a ‖ b	The ordered concatenation of the octet- or bit-strings a and b.
|L|	The length in bits of the octet- or bit-string L.
a ⊕ b	The bitwise exclusive-OR (bitwise addition modulo 2) of octet- or bit-strings a and b.
$x'H_1H_2\ldots H_{2n}H_{2n+1}$	An octet string represented as a sequence of Base16 digits (0-9, A-F). Each octet s_i in an n-octet string $S = s_1\, s_2 \ldots s_n$ is represented by a pair of digits in the Base16 alphabet $H_{2i-1}H_{2i}$ such that $s_i = H_{2i-1} \times 16 + H_{2i}$. See also BASE16() in section 5.3. For example, x’012345 is a sequence of octets 0x01, 0x23, 0x45.
BitLen(x)	Length in bits of bit string or octet string ***x***.
OctetLen(x)	Length in octets of octet string ***x***.

4 Key Management Process

4.1 Setup process for SM Manufacturers and KMCs

The SM Manufacturer Setup process (Step 1 in Figure 1; also section 8) is performed once when an SM Manufacturer adopts this key management specification, and infrequently thereafter (whenever the manufacturer's digital signing key pair expires, typically every 3 to 5 years).

The KMC Initialisation process (Step 2; also section 10) is performed once when a KMC is first commissioned, and infrequently thereafter (whenever the KMC's key establishment key pair expires, typically every 2 to 3 years).

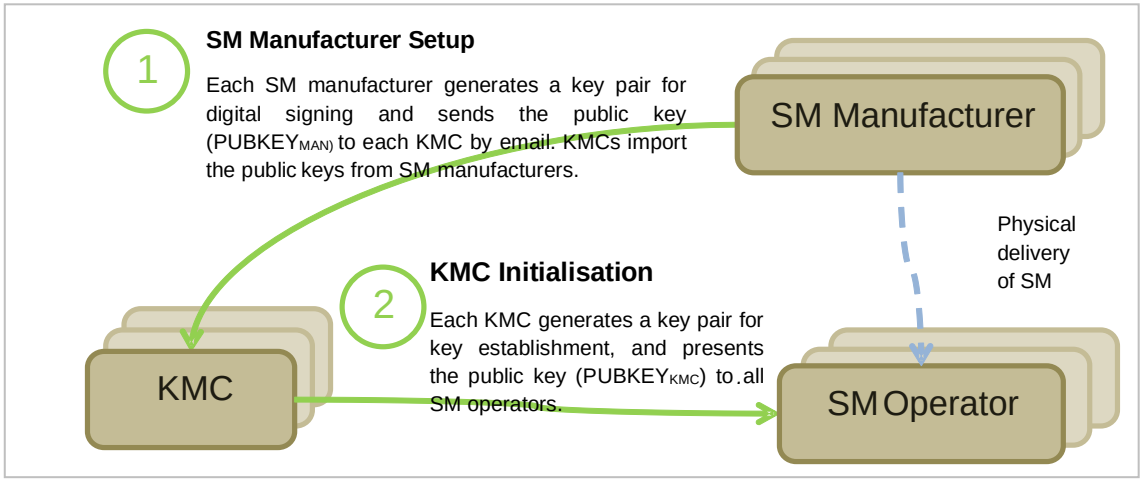

Figure 1 - SM Manufacturer Setup Process

4.2 Key publication after SM manufacture or maintenance

Whenever fresh cryptographic trust must be established in a Security Module (SM) – such as after manufacture, refurbishment or maintenance – the SM Initialisation process (Steps 3, 4 and 5 in Figure 2) must be performed.

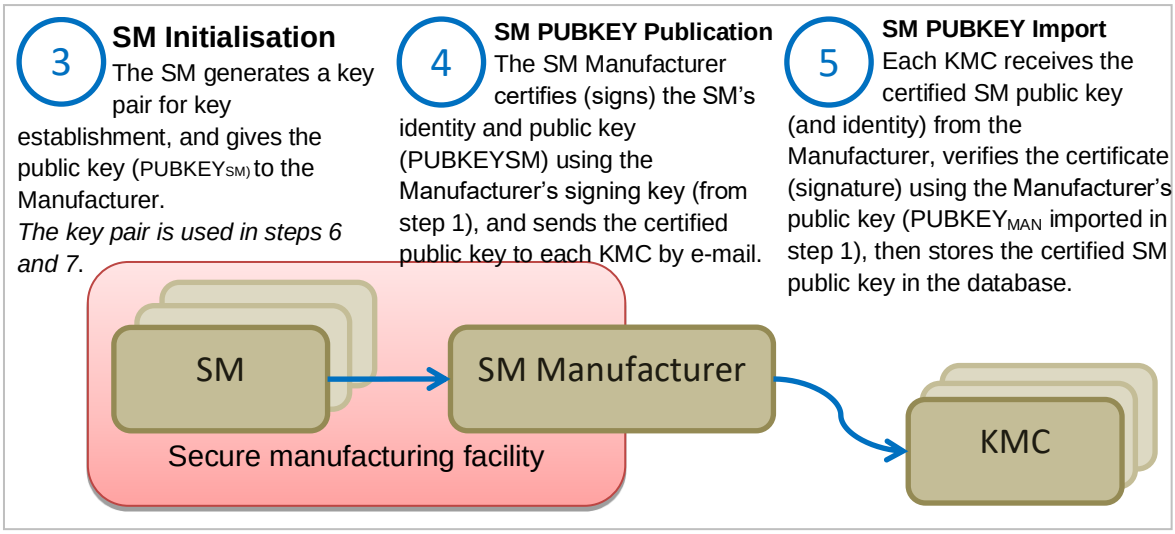

Figure 2 – SM initialisation Process

4.3 Vending Key Load Request and Response

Whenever an SM needs new or updated Vending Keys (VKs) from any KMC, the SM must prepare a Vending Key Load Request (Step 6 in Figure 3; also section 11) that is sent to the KMC. The KMC uses the SM's certified public key (imported in step 5) to verify that the request originated from an authentic SM, then replies with a Vending Key Load Response (Step 7 in Figure 3; also section 12) that authenticates the KMC to the SM and includes zero or more VKs.

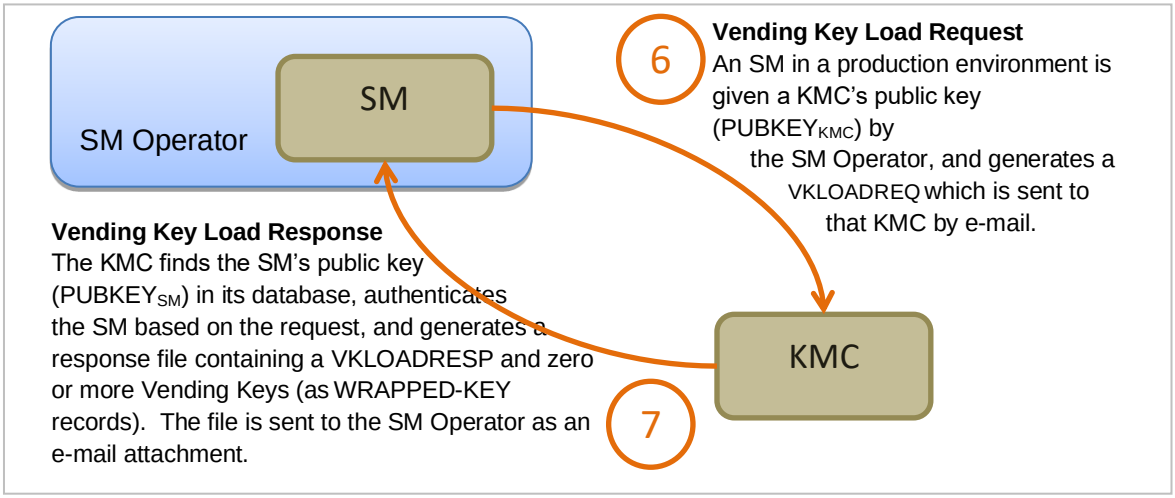

Figure 3 - Vending Key Load Request and Response

5 Data Types and Encodings

5.1 Types

Some data elements must be represented using a limited alphabet and/or a fixed size. This section specifies alphabets, size notations and encodings.

5.1.1 Alphabets

Table 1 names and describes various alphabets.

Table 1 - Alphabets

Alphabet	Short name	[POSIX RE]	Description
Printable ASCII	P	[\x20..\x7E] or [[:print:]]	Each octet is a single character in the US-ASCII [W:ASC] encoding, and SHALL be in the range of printable characters [W:ASC] (x'20 – x'7E inclusive).
Alphabetic (Letter)	A	[A-Za-z] or [[:alpha:]]	A printable ASCII character in the English alphabet, that is, a letter in the range 'A' (x'41) to 'Z' (x'5A) or 'a' (x'61) to 'z' (x'7A) inclusive.
Decimal	D	[0-9] or [[:digit:]]	A printable ASCII character in the range '0' (x'30) to '9' (x'39) inclusive, used as the alphabet for base 10 encoding.
Hexadecimal [W:HEX]	H	[0-9 A-F]	A printable ASCII character in the range '0' (x'30) to '9' (x'39) or 'A' (x'41) to 'F' (x'46) inclusive, used as the alphabet for Base16 encoding.
Alphanumeric	AN	[A-Za-z0-9]	A character that is either Alphabetic or Decimal.

5.1.2 Sizes

Table 2 gives a compact notation used to express fixed- or variable-length fields of a particular alphabet.

Table 2 - Field Notation

Notation	Description	Examples
nT	A fixed-length field of *n* characters from alphabet *T*. *n* is a decimal number, and *T* is a short name from section 5.1.1	1D, 3AN, 8H

Notation	Description	Examples
n-mT	A variable-length field of characters from alphabet T, with a minimum length of n and a maximum length of m characters. n and m are decimal numbers, and T is a short name from section 5.1.1.	2-4D, 0-16A
*x*n*T*	A variable-length field of characters from alphabet T, with length a multiple of x (a decimal number). T is a short name from section 5.1.1 and n is a literal 'n'.	2nH

5.1.3 IDENT

An IDENT is a special type comprising one Alphanumeric character (1AN) followed by zero to ninety-eight characters that are either Alphanumeric or one of the following: underscore ('_', x'5F), hyphen ('-', x'2D), period ('.', x'2E) or comma (',', x'2C). IDENT is described by the regular expression [POSIX RE] [A-Za-z0-9][A-Za-z0-9_\-.,]{0,39} (maximum length 40 characters).

5.1.4 TIMESTAMP

A TIMESTAMP is an instant in Coordinated Universal Time (UTC) represented as a complete date and time point using [ISO 8601] basic format: *YYYYMMDD***T***hhmmss***Z**.

The "T" and "Z" are literal, indicating a timestamp ("T") in UTC ("Z") format. All characters Y, M, D, h, m, s are Decimal . *YYYY* is a calendar year, *MM* is a calendar month (Jan = "01"), *DD* is a calendar day of the month (the first day is "01"). *hh* is an hour of day in the range "00" to "23", *mm* is a minute from "00" to "59", and *ss* is a second from "00" to "59".

Note that the use of *hh*="24" for midnight or *ss*="60" for a leap second are prohibited. No alternative representations, extended formats or separators are permitted.

5.2 BCD

Packed Binary Coded Decimal [W:BCD] is a compact representation for strings of decimal digits. Each digit is encoded as one nibble (4 bits) in the output.

Function description:

- **BCD(X)** outputs the packed Binary Coded Decimal representation of 2*n*-character decimal string X (type 2nD). If $X=d_1d_2\ldots d_{2n}$ where d_i is an octet in the decimal alphabet then **BCD(X)** outputs $x'd_1d_2\ldots d_{2n-1}d_{2n}$.

Example: If X="012345" with **OctetLen(**X**)**=6, then **BCD(**X**)**=x'012345 with **OctetLen(BCD(**X**))**=3.

5.3 BASE16 and BASE16-DECODE

The Base 16 encoding [RFC 4648] - section 8 is intended to represent arbitrary octet strings in the form of hexadecimal [W:HEX] strings.

Function description:

- **BASE16(X)** where X is a sequence of octets x_1 x_2 … x_n (an octet string) outputs a sequence of hexadecimal characters h_1 h_2 … h_{2n+1} (also an octet string) such that for every x_i (i = 1, 2, …, n), h_{2i-1} is the top 4 bits of x_i translated into the Hexadecimal alphabet and h_{2i} is the same translation of the bottom 4 bits of x_i.
- **BASE16-DECODE(X)** is the inverse of BASE16(X), also known as the decode operation:
  BASE16-DECODE(BASE16(*X*)) = ***X***. The operation SHALL fail if ***X*** is not a string in the Hexadecimal alphabet.

Note that in keeping with the definition of the Hexadecimal alphabet in section 5.1.1, only uppercase alphabetic characters ‘A’ to ‘F’ are permitted in the output.

5.4 Integer, field element and point conversions

The functions described here are defined in [ISO 18033-2] - sections 5.2.5, 5.3.1 and 5.4.3, [ANSI X9.62] section A.5, [ANSI X9.63] section 4.3 and [NIST SP800-56A] Appendix B.

A field element of the prime field Fq is an integer in the range [0, q-1], and may also be represented by a big endian octet string of length exactly $\lceil \log_2(q) / 8 \rceil$ octets. For the NIST P-384 elliptic curve q is a 384-bit integer.

A point P on an elliptic curve over Fq has coordinates (x_P, y_P) that are both field elements of Fq. Point P may be represented by an ordered concatenation of a prefix octet x’04 and the octet string representations of x_P and y_P. This specification permits only the uncompressed form for elliptic curve points (thus the prefix octet is x’04 as for field “H” in [ISO 18033-2] section 5.4.3 and the “PC” octet in [ANSI X9.63] section 4.3.6).

Within the scope of this specification all points SHALL be on the NIST P-384 elliptic curve (see section 6.5), and field elements have corresponding limitations.

5.4.1 Integer-to-Octet-String (I2OSP)

Integer-to-Octet-String(*x, L*) accepts an integer ***x*** in the range [0, ***L***-1] and outputs the octet string representation of ***x***. Conversion fails if ***x*** is outside the range [0, ***L***-1].

Let ***len***=$\lceil \log_2(\boldsymbol{L}) / 8 \rceil$, then output ***S*** where ***S*** is the string of octets S_1 S_2 … $S_{\mathbf{len}}$ satisfying $\boldsymbol{x} = \sum (2^{8(\text{len}-i)} \times S_i)$ for i = 1 to **len**.

5.4.2 Octet-String-to-Integer (OS2IP)

Octet-String-to-Integer(*S, L*) accepts a octet string ***S*** with length ***len***=$\lceil \log_2(\boldsymbol{L}) / 8 \rceil$ octets, and outputs the integer ***x*** represented by ***S***. Conversion fails if ***x*** is outside the range [0, ***L***-1].

Octet-String-to-Integer(Integer-to-Octet-String(*x, L*), L) = ***x*.**

5.4.3 Field-Element-to-Octet-String (FE2OSP)

Field-Element-to-Octet-String$_{Domain}$(*x*) is **Integer-to-Octet-String(*x*, *q*)** where ***q*** is given by the ***Domain*** parameters.

5.4.4 Octet-String-to-Field-Element (OS2FEP)

Octet-String-to-Field-Element$_{Domain}$(*S*) is **Octet-String-to-Integer(*S*, *q*)** where ***q*** is given by the ***Domain*** parameters.

Octet-String-to-Field-Element(Field-Element-to-Octet-String(*x*)) = ***x*.**

5.4.5 Point-to-Octet-String (EC2OSP)

Point-to-Octet-String$_{Domain}$(*P*) accepts a point $\boldsymbol{P} = (\boldsymbol{x_P},\boldsymbol{y_P})$ that is not the point at Infinity, and outputs the octet string $\boldsymbol{S}$ = x'04 ‖ **Field-Element-to-Octet-String$_{Domain}$($\boldsymbol{x_P}$)** ‖ **Field-Element-to-Octet-String$_{Domain}$($\boldsymbol{y_P}$)**.

5.4.6 Octet-String-to-Point (OS2ECP)

Octet-String-to-Point$_{Domain}$(*S*) accepts octet string ***S*** and outputs a point P constructed from the coordinates (field elements) $\boldsymbol{x_P}$ and $\boldsymbol{y_P}$ in Fq. Let ***FELen*** = $\lceil \log_2(\boldsymbol{q})/8 \rceil$ where ***q*** is given by the ***Domain*** parameters. Conversion fails if **OctetLen(*S*)** ≠ 1+2.***FELen***. Interpret ***S*** as an ordered concatenation of fixed-length octet strings ***PC*** (1 octet), $\boldsymbol{S_L}$ (***FELen*** octets) and $\boldsymbol{S_R}$ (***FELen*** octets). FAIL if ***PC*** ≠ x'04. Compute $\boldsymbol{x_P}$ = **Octet-String-to-Field-Element$_{Domain}$($\boldsymbol{S_L}$)** and $\boldsymbol{y_P}$ = **Octet-String-to-Field-Element$_{Domain}$($\boldsymbol{S_R}$)**, then output point $\boldsymbol{P} = (\boldsymbol{x_P},\boldsymbol{y_P})$. Point P is not guaranteed to be a valid point on the curve.

Octet-String-to-Point(Point-to-Octet-String(*P*)) = ***P*.**

5.5 CRC16-MODBUS

The 16-bit checksum specified in [IEC 62055-41] - section 6.3.7, also known as CRC16/MODBUS [CRC-CAT].

Function description:

- **CRC16-MODBUS(*x*)** computes and outputs a 16-bit checksum over the input octet string ***x*** using a Cyclic Redundancy Code with the generator polynomial ($x^{16} + x^{15} + x^2 + 1$) and the initial checksum 0xFFFF.

This specification treats the CRC as an integer (16-bit big endian bit string), whereas [IEC 62055-41] formats the CRC as a two octet little-endian value.

The CRC parameters in the Rocksoft™ model [ROCKSOFT] are:

width	16
poly	0x8005
init	0xffff
refin	True
refout	True
xorout	0x0000

check	0x4b37 (input="123456789")

Lammert's On-line CRC calculator [LAMMERT] supports this checksum and can be used to validate implementations.

5.6 LVCONCAT

(Mnemonic: "Length-Value Concatenation") LVCONCAT is a formatting function that produces an ordered concatenation of octet strings each with a length prefix. The output is a one-to-one mapping of the inputs, can be parsed unambiguously into the original inputs, and is prefix-free.

LVCONCAT is designed to format input fields to cryptographic functions such as key derivation and message authentication. It follows the principles of [CM10] to avoid exploitable ambiguities in interpretation, and meets the requirements for the following input data fields:

- *FixedInput* for the KDF in [NIST SP800-108] (sections 5 and 7).
- *SharedInfo* for the KDF in [ANSI X9.63] - section 8, equivalent to *OtherInfo* in [NIST SP800-56A] - section 5.8.1.
- *MacData* (also called *M*) for key confirmation in [ISO 11770-3] - section 9, [ANSI X9.63] and [NIST SP800-56A] - section 8.2.

Function description:

- **LVCONCAT(I_1, I_2, …, I_n)** outputs as an octet string a one-to-one prefix-free encoding of input octet strings $I_1, I_2, \ldots, I_n$ ($0 \le n < 256$, **OctetLen(I_i)**<256) that can be unambiguously parsed into the original inputs.

Input:

- $I_1, I_2, \ldots, I_n$, the n input octet strings.

Process:

- If $n > 255$ then FAIL.
- Set S to a 1 octet (8-bit) integer representation of n.
- For $j = 1, 2, \ldots, n$ do:
  - If **OctetLen(I_j)** > 255 then FAIL.
  - Set L to a 1 octet (8-bit) integer representation of **OctetLen(I_j)**.
  - $S = S \| L \| I_j$.
- Output S.

5.7 Delimited Field strings

A delimited string is an ordered concatenation of fields that are separated from each other by a non-alphabetic delimiter (a character outside the field alphabet).

DFCONCAT *(mnemonic: "Delimited Field Concatenation")* is a formatting function that produces an ordered concatenation of printable ASCII strings that are separated from each other by a delimiter from the printable ASCII alphabet. The output is prefix-free with respect to all other outputs for the same number of input fields.

DFPARSE is the corresponding parsing function.

5.7.1 DFCONCAT

Function description:

- **DFCONCAT(*DELIM*, I_1, I_2, …, I_n)** outputs as a printable ASCII string a one-to-one encoding of the input printable ASCII strings I_1, I_2, …, I_n none of which may contain the printable ASCII character ***DELIM***. The output can be unambiguously parsed into the original inputs.

Input:

- ***DELIM***, the delimiter character (printable ASCII, 1P).
- I_1, I_2, …, I_n, the ***n*** input printable ASCII strings.

Process:

- If ***DELIM*** is not a printable ASCII character (1P) then FAIL.
- Set ***S*** to an empty octet string.
- For ***j*** = 1, 2, …, ***n*** do:
  - If any octet in I_j equals ***DELIM*** or is not printable ASCII then FAIL.
  - ***S*** = ***S*** ‖ I_j ‖ ***DELIM***.
- Output ***S***.

Note that the output always ends with the delimiter character DELIM; this is necessary to obtain a prefix-free encoding of the inputs.

5.7.2 DFPARSE

Function description:

- **DFPARSE(*DELIM*, *S*)** is the inverse of **DFCONCAT**, also known as the parsing operation: **DFPARSE(*DELIM*,DFCONCAT(*DELIM*, I_1, I_2, …, I_n)) = I_1, I_2, …, I_n**. The operation fails if octet string ***S*** is not a valid output of **DFCONCAT(*DELIM*, …)** (that is, a string of printable ASCII characters ending in ***DELIM***).

Input:

- ***DELIM***, the delimiter character (printable ASCII, 1P).
- ***S***, the octet string to be parsed.

Process:

- If ***DELIM*** is not a printable ASCII character (1P) then FAIL.
- If the last octet in ***S*** is not ***DELIM*** then FAIL(*"Bad encoding in input"*).

- If any octet in ***S*** is not printable ASCII then FAIL(*"Bad characters in input"*).
- Split ***S*** into fields O_1, O_2, …, O_n delimited by the character ***DELIM***.
- Output ***n*** and O_1, O_2, …, O_n.

5.8 Records

A record is a data structure with multiple fields and a printable ASCII representation. Within this specification all data transfer and storage formats are defined as records.

A record combines into a printable ASCII string: a type indicator, an ordered sequence of fields, and a checksum. The type indicator must be an IDENT, and each field must be a printable ASCII string that does not contain the delimiter character.

5.8.1 BUILD-RECORD

BUILD-RECORD(*rectype*, delim, *n*, I_1, I_2, …, I_n) accepts as input an IDENT ***rectype***, a single printable ASCII character **delim** that SHALL NOT be alphanumeric, a positive integer ***n*** (0 < ***n*** < 256), and ***n*** printable ASCII strings that SHALL NOT contain the delimiter character ***delim***. This function constructs ***R*** = **DFCONCAT(*delim*, *rectype*, I_1, I_2, …, I_n)**, computes ***C*** = **CRC16-MODBUS(R)** (C is a 16-bit big endian bit string) and outputs ***R*** ‖ **BASE16(*C*)**.

5.8.2 PARSE-RECORD

PARSE-RECORD(*rectype*, delim, *n*, *S*) accepts as input an IDENT ***rectype***, a single printable ASCII character **delim** that SHALL NOT be alphanumeric, a positive integer ***n*** (0 < ***n*** < 256), and an octet string ***S***. This function verifies that **S** is a record of type **rectype** and has a valid CRC.

Process:

- If ***S*** does not start with the string ***rectype*** ‖ ***delim*** then FAIL(*"Input is not a record of type "* ‖ **rectype**).
- If **OctetLen(*S*)** < (**OctetLen(*rectype*)**+5) then FAIL(*"Missing CRC on record "* ‖ **rectype**).
- Split ***S*** into ***R*** ‖ ***C'***, where ***C'*** is the last 4 characters of S.
- Compute ***C*** = **CRC16-MODBUS(*R*)**, ***C*** is a 16-bit big endian bit string.
- If ***C'*** ≠ **BASEI6(*C*)** then FAIL(*"Bad checksum on record "* ‖ **rectype**).
- Parse ***R*** using **DFPARSE(*delim*, *R*)** to recover fields O_1, O_2, …, O_m. Propagate errors.
- If ***n*** ≠ ***m*** then FAIL(*"Wrong number of fields in record "* ‖ **rectype**).
- Output O_1, O_2, …, O_n.

6 Cryptographic Primitives

6.1 AES-192 in CCM mode

The AES block cipher with 192-bit cipher key as specified in [ISO 18033-3] and [FIPS PUB 197], operated in CCM mode as specified in [ISO 19772] and [NIST SP800-38C] and [RFC 3610], with a tag length (MAC) of 128 bits.

This specification requires that the CCM implementation SHALL use a Flags Octet value of x'7B in B_0, and SHALL NOT permit or accept any other value for the Flags Octet.

Function description:

- **AES-192-CCM$_{ENC}$(*K*, *N*, *additional*, *plaintext*)** computes a 128-bit keyed authentication tag over the octet string inputs ***plaintext*** (maximum length 2^{23}-1 octets) and ***additional*** (maximum length 2^{23}-1 octets) using the 192-bit key ***K*** and 96-bit nonce ***N***, enciphers the input ***plaintext*** using ***K*** and ***N***, and outputs a ciphertext that includes the authentication tag.
- **AES-192-CCM$_{DEC}$(*K*, *N*, *additional*, *ciphertext*)** accepts octet string inputs ***ciphertext*** (maximum length 2^{23}+15 octets) and ***additional*** (maximum length 2^{23}-1 octets) where ***ciphertext*** includes a 128-bit keyed authentication tag, deciphers the ***ciphertext*** using the 192-bit key ***K*** and 96-bit nonce ***N*** to produce ***plaintext***, verifies the authentication tag over ***plaintext*** and ***additional*** using ***K*** and **N**, and outputs ***plaintext***.

6.2 SHA-384

The SHA-384 hash function, as specified in [ISO 10118-3] and [FIPS PUB 180-4].

Function description:

- **SHA-384(*X*)** outputs a 384-bit digest computed over the input bit string ***X***.

6.3 HMAC-SHA-384-192

HMAC-SHA-384-192 is defined in RFC 4868 as the keyed-hash message authentication code (HMAC) specified in [ISO 9797-2] and [FIPS PUB 198-1] and [RFC 2104], using the hash function SHA-384 (section 6.2), with the MAC truncated to the leftmost 192 bits.

Within the scope of this specification HMAC-SHA-384-192 is only used with a key of less than 1024 bits (the block size of SHA-384). The implementation given below has been simplified accordingly.

Function description:

- **HMAC-SHA-384-192(*K*, *text*)** outputs a 192-bit message authentication code (MAC) computed over the *n*-octet input ***text*** ($0 \le n < 2^{16}$) using the *m*-octet key ***K*** ($0 < m < 128$).

Input:

- ***K***, a secret key (as an octet string).

- text, the data on which the HMAC is computed.

Process:

- ***B*** = 128, an integer constant giving the block size in octets of the hash function (SHA-384).
- ***ipad*** = x'3636…, an octet string constant of length B (the octet x'36 repeated B times).
- ***opad*** = x'5C5C…, an octet string constant of length B (the octet x'5C repeated B times).
- If **OctetLen(*K*)** ≥ **B** then FAIL.
- If **OctetLen(*text*)** ≥ 2^{16} then FAIL.
- Append zeros (octets x'00) to the end of key ***K*** to create a ***B***-octet string K_0.
- Compute ***MAC*** = **SHA-384(** ($K_0 \oplus$ ***opad***) ‖ **SHA-384(** ($K_0 \oplus$ ***ipad***) ‖ ***text*** **))**.
- Output the leftmost 192 bits of ***MAC***.

6.4 KDF-X963-SHA-384

The Key Derivation Function (KDF) specified in section 5.6.3 of [ANSI X9.63], [ISO 11770-3] - Annex B.3, and [SEC 1] section 3.6.1, using the hash function SHA-384 (section 6.2).

Within the scope of this specification **KDF-X963-SHA-384** is only used with ***keydatalen*** = 384 bits, and small ***Z*** and ***SharedInfo*** (less than 2^{19} bits). The implementation given below has been simplified accordingly.

Function description:

- **KDF-X963-SHA-384(*Z*, *SharedInfo*, *keydatalen*)** outputs a ***keydatalen***-bit key derived from an asymmetrically shared secret ***Z*** (maximum length 2^{10}-1 bits) and octet string ***SharedInfo*** (maximum length 2^{16}-1 octets).

Input:

- **Z**, a bit string of secret data (maximum length 2^{10}-1 bits).
- ***SharedInfo***, an octet string of non-secret data, 0 < **OctetLen(*SharedInfo*)** ≤ (2^{16}-1).
- ***keydatalen***, an integer giving the length in bits of keying data to be generated.

Process:

- ***hashlen*** = 384, an constant integer giving the length in bits of the digest (output) produced by the hash function (**SHA-384**).
- If **BitLen(*Z*)** ≥ 2^{10} then FAIL.
- If **OctetLen(*SharedInfo*)** ≥ 2^{16} then FAIL.
- If ***keydatalen*** > ***hashlen*** then FAIL.

- Set ***counter*** (a 32-bit, big-endian bit string) to x'00000001.
- Compute ***KeyData*** = **SHA-384(*Z* ‖ *counter* ‖ *SharedInfo*)**.
- Output the leftmost ***keydatalen*** bits of ***KeyData***.

6.5 ECC CDH in NIST P-384

The Cofactor Diffie-Hellman (CDH) primitive specified in [ISO 11770-3] - Annex D, [ANSI X9.63] section 5.4.2 ("Modified Diffie-Hellman Primitive"), [NIST SP800-56A] section 5.7.1.2 and [SEC 1] section 3.3.2.

This specification requires that all CDH operations SHALL be performed using the NIST P-384 curve and domain parameters that are specified in [FIPS PUB 186-3], [ANSI X9.62] (as "ansix9p384r1") and [SEC 2] (as "secp384r1"). Octet string representations of points on the elliptic curve SHALL use uncompressed form affine coordinates ([ISO 18033-2] - section 5.4.3, [ANSI X9.63] section 4.3.6) as described by the Point-to-Octet-String conversion (section 5.4.5).

CDH uses scalar (integer) multiplication on an elliptic curve over a finite prime field, as defined in [ISO 15946-1] - section A.1.2 and A.4, and in [SEC 1].

Function description:

- **ECC-CDH$_{Domain}$(d_A, Q_B)** accepts A's private key d_A (an integer in the range [1,n-1] where n is given by the ***Domain*** parameters, always NIST P-384 in this specification) and B's public key Q_B (a point on the elliptic curve), and computes and outputs a shared secret octet string Z.

Input:

- d_A, the private key of entity A (an integer in the range [1,n-1]).
- Q_B, the public key of entity B (a point on the curve).

Process:

- Use domain parameters (q, FR, a, b, G, n, h) = ***NIST P-384***.
- Compute the point $P = (x_P, y_P) = \mathbf{h}\, d_A\, Q_B$ (scalar multiplication of a point on an elliptic curve).
- If P is the point at infinity then FAIL.
- Set Z to x_P (the x-coordinate of P).
- Zeroise intermediate results and output **Field-Element-to-Octet-String(Z)**.

6.6 One-Pass Unified Model Key Agreement Scheme C(1, 2, ECC CDH)

The One-Pass Unified Model key agreement scheme using the Elliptic Curve Cofactor Diffie-Hellman (ECC CDH) primitive, also known as C(1, 2, ECC CDH) or C(1e, 2s). The scheme is specified in [NIST SP800-56A] - section 6.2.1.2 and [ANSI X9.63] section 6.5, and is used with bilateral key confirmation.

The scheme is a composite of [ISO 11770-3] key agreement mechanisms 1 and 2 and thus complies with that standard although it is not specifically identified and described.

Key confirmation from the SM to the KMC is modified to use a Time Variant Parameter (TVP) instead of a random nonce, allowing the confirmation to be included in the first protocol message, but slightly reducing freshness guarantees. This modification is permitted (the Nonce used in key confirmation is not required to be random) and the first protocol message is consistent with the entity authentication requirements of [ISO 9798-4].

The scheme is not detailed here; instead the scheme steps and all procedural prerequisites are included in the Vending Key Load Request (section 11), Vending Key Load Response (section 12) and KEK Confirmation (section 13) processes.

6.7 ECDSA in NIST P-384

The Elliptic Curve Digital Signature Algorithm (ECDSA) specified in [ISO 14888-3], [ANSI X9.62], [FIPS PUB 186-3] and [SEC 1].

This specification requires that all ECDSA operations SHALL BE performed using the NIST P-384 curve and domain parameters that are specified in [FIPS PUB 186-3], and that the hash function SHALL BE SHA-384 (section 6.2).

6.7.1 ECDSA-SIGN

ECDSA-SIGN$_{Domain,Hash}$(d_A, M) accepts A's private key d_A (an integer in the range [1,n-1] where n is given by the ***Domain*** parameters) and a message ***M*** (an octet string), and computes and outputs a signature **(r, s)** where ***r***, ***s*** are both in [1,n-1].

In this specification the Domain is always NIST P-384, and the Hash function is SHA-384.

6.7.2 ECDSA-VERIFY

ECDSA-VERIFY$_{Domain,Hash}$(Q_A, M, *(r, s)*) accepts A's public key Q_A (a point on the elliptic curve given by the ***Domain*** parameters), a message ***M*** (an octet string), and a purported signature ***(r, s)***; checks the purported signature and outputs an indication of whether the signature is valid or not: "valid" or "invalid".

In this specification the Domain is always NIST P-384, and the Hash function is SHA-384.

6.8 GENERATE-KEY

The Elliptic Curve key generation primitive specified in [ISO 15946-1] - section 6.1, [ANSI X9.63] section 5.2.1 and [FIPS PUB 186-3] section B.4 (using candidate testing).

This specification requires that all CDH and ECDSA keys use the NIST P-384 curve and domain parameters; see sections 6.5 and 6.7.

Function description:

- **GENERATE-KEY()** generates and outputs a random private key $\boldsymbol{d_A}$ (an integer in the range [1,$\boldsymbol{n}$-1] where $\boldsymbol{n}$ is given by NIST P-384) and the corresponding public key $\boldsymbol{Q_A}$ (a point on the P-384 curve).

Process:

- Use domain parameters (q, FR, a, b, G, **n**, h) = ***NIST P-384***.
- Select a unique and unpredictable integer $\boldsymbol{d_A}$ in the range [1, **n**-1]:
  - Obtain a string ***S*** of 384 bits from a random bit generator (RBG) with a security-strength of 192 bits or more.
  - Set ***I*** = **Octet-String-to-Field-Element(*S*)**.
  - If (***I*** > **n** – 2) then discard **S** and ***I*** and repeat the generation.
  - Set $\boldsymbol{d_A} = \boldsymbol{I} + 1$.
- Compute the public key $\boldsymbol{Q_A} = \boldsymbol{d_A}\ \boldsymbol{G}$ (scalar multiplication of a point on an elliptic curve).
- Output the key pair $\mathbf{d_A}$ and $\mathbf{Q_A}$.

6.9 VALIDATE-KEY

The public key validation primitive specified in [ISO 15946-1] - section 7, [ANSI X9.63] section 5.2.2.1 (“Standard Public Key Validation Primitive”), and [NIST SP800-56A] section 5.6.2.5 (“ECC Full Public Key Validation Routine”).

Function description:

- **VALIDATE-KEY($\boldsymbol{Q_B}$)** outputs TRUE if $\boldsymbol{Q_B} = (x_Q, y_Q)$ is a point on the NIST P-384 curve and is not the identity element, or fails otherwise.

Process:

- Use domain parameters (**q**, FR, a, b, G, **n**, h) = ***NIST P-384***.
- If $\boldsymbol{Q_B}$ is the point at Infinity then FAIL.
- If x_Q is not in the range [0, **q**-1] or y_Q is not in the range [0, **q**-1] then FAIL.
- Verify that $(y_Q)^2 \equiv (x_Q)^3 + a.x_Q + b \pmod q$ or FAIL.
- Verify that $\boldsymbol{P} = \boldsymbol{n}\ \boldsymbol{Q}$ (scalar multiplication) is the point at Infinity or FAIL.
- Output TRUE.

7 Data Formats and Structures

7.1 PKID

Rectype = “SMID.1” or “SMMAN.1” or “KMCID.1” Delim = ‘:’ = x’3A

A $PKID_A$ is a record (section 5.8) with delimiter ‘:’ that identifies entity A by binding together the unique name of entity A with A’s public key:

- The tuple (Manufacturer, MID) uniquely identifies entity A.
- The tuple (Manufacturer, MID, GNT) uniquely identifies a public key associated with entity A.
- Given $PKID_A$ it is difficult to find a public key $Q_A' \neq Q_A$ that satisfies the Fingerprint.

The record type indicates the role of the entity in the key management infrastructure:

- *rectype* = “**SMID.1**” if entity A is an SM;
- *rectype* = “**SMMAN.1**” if A is an SM Manufacturer;
- *rectype* = “**KMCID.1**” if A is a KMC.

The record contains the following fields, in order:

Position	Field	Type	Description
1	Manufacturer	IDENT	Identifies the manufacturer of entity A.
2	MID	IDENT	A unique Module IDentifier of entity A with respect to the Manufacturer; the tuple (Manufacturer, MID) must be globally unique.
3	GNT	TIMESTAMP	“Generation time”: the time at which A’s key pair (***d*****$_A$**, ***Q*****$_A$**) was generated. The key pair SHALL NOT be used for signing or key agreement before this date.
4	Fingerprint	16H	A collision resistant hash that binds the preceding fields and record type to A’s public key Q_A. Let ***S*** = **DFCONCAT(**‘:’, ***rectype***, ***Manufacturer***, ***MID***, ***GNT***, **Q_AHEX)** where **Q_AHEX** = **BASE16**(**Point-To-Octet-String**(**Q_A**), then ***Fingerprint*** is the leftmost 16 characters of **BASE16(SHA-384(*S*))**.

To verify the *Fingerprint* of a $PKID_A$ and purported public key Q_A': parse $PKID_A$ using PARSE-RECORD() and compute Fingerprint' using the recovered fields and Q_A', then compare the recovered *Fingerprint* with the computed Fingerprint'.

7.2 PUBKEY

Rectype = “PK.ECDH.1” or “PK.ECDSA.1” Delim = ‘|’ = x’7C

A $PUBKEY_A$ is public key certificate [W:CERT] that identifies entity A and contains A’s public key. The certificate may be signed by an Issuer, self-signed, or unsigned.

The PUBKEY is represented as a record (section 5.8) with delimiter ‘|’, and the record type indicates the purpose and permitted usage of the public key:

- *rectype* = “**PK.ECDH.1**” for a ECC Cofactor Diffie Hellman (section 6.5) public key that is reserved for use in the key management processes specified in this document;
- *rectype* = “**PK.ECDSA.1**” for an ECDSA public key with NIST P-384 domain parameters.

The record contains the following fields, in order:

Position	Field	Type	Description
1	Subject (ID_A)	Printable	PKID (section 7.1) of the owner of the public key Q_A; includes the public key Fingerprint.
2	Q_AHEX	194H	Entity A’s public key $\boldsymbol{Q_A}$, encoded as **BASE16(Point-To-Octet-String($\boldsymbol{Q_A}$))**.
3	Expiry	TIMESTAMP	The time at which A’s key pair ($\boldsymbol{d_A}$, $\boldsymbol{Q_A}$) expires. Expiry SHALL be greater than the GNT (generation time) field of the Subject. An expired key pair SHALL NOT be used for signing or for key agreement, although an expired ECDSA key may be used to verify signatures created before the expiry date.
4	Issuer	∅ or Printable	PKID of the Issuer responsible for generating the Signature. Leave empty if the PUBKEY is unsigned.
5	Signature	∅ or 192H	A digital signature that binds together the preceding fields and record type, or empty if the PUBKEY is unsigned. Let $\boldsymbol{M}$ = **DFCONCAT(**‘|’, **rectype**, **Subject**, **Q_AHEX**, **Expiry)**, and let $\boldsymbol{(r, s)}$ = **ECDSA-SIGN($\boldsymbol{d_{ISSUER}}$, $\boldsymbol{M}$)** where $\boldsymbol{d_{ISSUER}}$ is the Issuer’s private key, then the Signature is **BASE16(Integer-to-Octet-String($\boldsymbol{r}$, $\boldsymbol{n}$)** ‖ **Integer-to-Octet-String($\boldsymbol{s}$, $\boldsymbol{n}$))** where $\boldsymbol{n}$ is given by NIST P-384.

To verify the *Signature* of a $PUBKEY_A$: parse $PUBKEY_A$ using PARSE-RECORD(), construct M (as described for the *Signature* field) and verify *Signature* using ECDSA-VERIFY(Q_{ISSUER}, M, *Signature*), where Q_{ISSUER} is the Issuer’s public key.

7.3 VKLOADREQ

Rectype = “VKLOAD.REQ.1”
Delim = ‘|’ = x’7C

A Vending Key Load Request $VKLOADREQ_{SM}$ is a record (section 5.8) of type “VKLOAD.REQ.1” with delimiter ‘|’ that is constructed by the SM and sent to the KMC to request vending keys.

The record contains the following fields, in order:

Position	Field	Type	Description
1	ID_{SM}	Printable	PKID (section 7.1) of the requesting SM.
2	ID_{KMC}	Printable	PKID (section 7.1) of the target KMC.
3	TVP_{KMC}	TIMESTAMP	Time variant parameter taken from the SM’s RTC.
4	HWID	IDENT	SM hardware model and revision (see section 9.1).
5	FWID	IDENT	SM firmware application and version (see section 9.1).
6	Q_EHEX	194H	SM ephemeral public key $\boldsymbol{Q_E}$ (see section 11) encoded as **BASE16(Point-To-Octet-String($\boldsymbol{Q_E}$)).**
7	$MacTag_{SM}$HEX	48H	SM key confirmation $\boldsymbol{MacTag_{SM}}$ (see section 11) encoded as **BASE16($\boldsymbol{MacTag_{SM}}$)**.

7.4 VKLOADRESP

Rectype = “VKLOAD.RESP.1”
Delim = ‘|’ = x’7C

A Vending Key Load Response $VKLOADRESP_{KMC}$ is a record (section 5.8) of type “VKLOAD.RESP.1” with delimiter ‘|’ that is constructed by the KMC and sent to the SM in response to a successful VKLOADREQ.

The record contains the following fields, in order:

Position	Field	Type	Description
1	ID_{KMC}	Printable	PKID (section 7.1) of the responding KMC.
2	ID_{SM}	Printable	PKID (section 7.1) of the requesting SM.
3	TVP_{KMC}	TIMESTAMP	Time variant parameter copied from the SM’s $VKLOADREQ_{SM}$.
4	$MacTag_{KMC}$HEX	48H	KMC key confirmation $\boldsymbol{MacTag_{KMC}}$ (see section 12) encoded as **BASE16($\boldsymbol{MacTag_{KMC}}$)**.

7.5 WRAPPED-KEY

Rectype = “KEY.1” Delim = ‘|’ = x’7C

A Wrapped Key is a record (section 5.8) of type “KEY.1” with delimiter ‘|’ that is constructed by the KMC and sent to the SM. This constitutes a symmetric key transfer scheme consistent with [ISO 11770-2] mechanism 2.

The record contains the following fields, in order:

Position	Field	Type	Description
1	Nonce	24H	A 96-bit value represented in the Hexadecimal alphabet. Each WRAPPED-KEY under a specific KEK must have a unique nonce.
2	Attributes	P	The attributes associated with the key, encoded as a delimited printable ASCII string using a card format as described in section 7.5.1 below. Supported attributes are defined in Appendix B – Vending Key attributes.
3	ProtectedKey	H	The key material ***K*** (maximum length 160 bits) protected under the Key Exchange Key (KEK) using authenticated encryption (with ***Attributes*** as associated data) and encoded in the Hexadecimal alphabet. **ProtectedKey** = **BASE16(AES-CCM(*KEK*, *Nonce*, *Attributes*, *K*))**.

7.5.1 Attributes

Attributes are a collection of unique attribute names N_i (type 3AN) and corresponding values V_{Ni} (type P), $i = 1,2, \ldots,n$. The encoding, range and interpretation of V_{Ni} is determined by N_i according to the Vending Key attributes table given in Appendix B – Vending Key attributes, but in all cases V_{Ni} SHALL be printable ASCII (and SHALL exclude the record and field delimiter characters ‘|’=x’7C and ‘;’=x’3B) with a maximum length of 252 characters[1].

The Attributes field of a WRAPPED-KEY is encoded as a delimited printable ASCII string using a card format: each name N_i and associated value V_{Ni} is concatenated to form a single card. Let $S_1,\ldots,S_n$ be the names $N_1,\ldots,N_n$ sorted in strictly ascending lexicographical order [W:LEX] in the ASCII alphabet (no duplicates are permitted), then ***Attributes*** = ***DFCONCAT(***‘;’, $S_1 \| V_{S1}$, …, $S_n \| V_{Sn}$**)**.

[1] The length limit of 252 characters ensures that each string $N_i \| V_{Ni}$ is a valid input field to LVCONCAT.

8 SM Manufacturer Setup

Prior to SM initialisation an SM Manufacturer SHALL:

- Select a unique name MANUFACTURER (an IDENT) to identify itself.
    - The STSA SHOULD provide a registry service for Manufacturer names.
- Generate an asymmetric digital signature key pair for the purpose of certifying SM public keys.
    - The key pair SHALL be an ECDSA key pair using the NIST P-384 domain parameters and having a security-strength of at least 192 bits, and SHALL be generated using an RBG having equivalent (or stronger) security-strength.
    - The key pair SHALL be generated and managed with respect to the principles of split knowledge and dual control. It SHALL NOT be possible for any single operator to sign an SM public key using the private digital signature key.
    - The secret key SHALL be protected by an HSM. The HSM SHALL meet the security prerequisites for a KMC HSM (section 10.1).
- Publish the self-signed public key to all KMCs.
    - The public key SHALL be published as a self-signed PUBKEY record (referred to as $PUBKEY_{MAN}$; see section 7.2) with record type “PK.ECDSA.1”.
    - The PUBKEY Expiry SHALL be set to the time of generation plus the Originator Usage Period (see below).
    - A procedure to publish $PUBKEY_{MAN}$ SHALL be specified by KMC standards or operational documentation. Each recipient of the $PUBKEY_{MAN}$ SHALL check that the public key is not expired, and SHALL check the validity of the public key by manually confirming the public key’s fingerprint over an independent communication channel.

The Manufacturer’s key pair SHALL have a lifespan (Originator Usage Period) of at most 3 years (consistent with [NIST SP800-57 PART 1]):

- When the Manufacturer’s key pair expires, the Manufacturer SHALL generate a new key pair and publish the public key in the manner prescribed by this section.
- The Manufacturer SHALL NOT certify SM public keys using an expired key.
    - The Manufacturer’s private key d_{MAN} SHALL be associated with an expiry date such that the signature operation SHALL NOT generate a signature using an expired key.
- A KMC SHALL NOT trust any $PUBKEY_{SM}$ for which the GNT (the point in time at which the SM key pair was generated) is more recent than the expiry date of the $PUBKEY_{MAN}$ used to certify $PUBKEY_{SM}$.

8.1 Recommended process to generate and publish $PUBKEY_{MAN}$

The following process is RECOMMENDED:

- The SM Manufacturer selects a unique name MANUFACTURER (an IDENT).
- The Manufacturer uses an HSM to generate and store a unique ECDSA key pair (d_{MAN}, Q_{MAN}) using the NIST P-384 domain parameters, in accordance with [ISO 14888-3], [FIPS PUB 186-3], [ANSI X9.62] and/or [SEC 1].
- The Manufacturer constructs a $PKID_{MAN}$ with *rectype* “SMMAN.1”, *MID* “A”, and *GNT* the time at which d_{MAN} was generated.
- The Manufacturer constructs a $PUBKEY_{MAN}$ with *rectype* “PK.ECDSA.1” and Subject $PKID_{MAN}$. Expiry is at most 3 years after the *GNT*. The *Issuer* is $PKID_{MAN}$ and the *Signature* is generated using d_{MAN}.
- On demand by any KMC, the Manufacturer sends to the KMC the $PUBKEY_{MAN}$ in record-in-email format (Appendix C – Record-in-email format).
- Operating under the principle of dual control, two KMC operators obtain the Fingerprint from the Manufacturer and confirm the Fingerprint in the $PKID_{MAN}$, then instruct their system to import and trust the $PUBKEY_{MAN}$.
    - If no prior relationship between the KMC and the Manufacturer exists, then the operators SHOULD obtain the Fingerprint via a face-to-face meeting with the Manufacturer, or through a Trusted Third Party. Where there is a prior relationship a telephonic confirmation of the Fingerprint is adequate.

9 SM Initialisation

9.1 Prerequisites: SM

An SM SHALL have:

- A high quality entropy source that has been assessed using statistical tests from NIST SP800-22. The SM SHALL implement a continuous quality test on the output of the entropy source, for example by ensuring that adjacent blocks read from the source are distinct.
- A deterministic Random Bit Generator (RBG) seeded from the entropy source, with a security-strength of 192 bits or more.
    - The RBG SHALL comply with [ISO 18031], [NIST SP800-90], [ANSI X9.82] and/or [SEC 1].
- A real-time clock (RTC) for which the state is protected within the SM's cryptographic boundary.
    - The RTC SHOULD NOT drift by more than 3 days over the documented lifetime or maintenance interval of the SM.
- Secure storage for sensitive data. All keys and sensitive data SHALL include integrity protection. Key separation and substitution prevention SHALL be assured (for example using techniques from [ISO 11568-2]).

  Keys and sensitive data may be stored using one of the following techniques:
    - Within the cryptographic boundary of the SM, in non-volatile memory that is erased on tamper, and SHALL include integrity protection (such as a checksum); *OR*
    - Authentically encrypted under a Storage Key that is securely stored using the previous technique.
- Tested implementations of all cryptographic primitives (section 6) required by this specification.
- An authentic copy of the NIST P-384 domain parameters.
- HWID (string of type IDENT), a hardware identifier that SHALL be composed of a MANUFACTURER, MODEL and REVISION.
- FWID (string of type IDENT), a firmware application and version identifier.
- MID (string of type IDENT), a unique hardware identifier or assigned soft identifier. Each device shall have a MANUFACTURER-unique MID, not merely a MODEL-unique MID. A model name or code can be used as a MID prefix to guarantee this.

An SM SHOULD comply with a recognised standard for cryptograph modules such as [ISO 19790], [FIPS PUB 140-2], or [PCI HSM]. The target security level or evaluation criteria for such compliance are beyond the scope of this document. The STSA SHOULD maintain a Code of Practice detailing the security requirements for an SM.

9.2 SM Initialisation and PUBKEY certification

Secure key agreement between an SM and a KMC – including authentication of the SM – requires that the SM contain secret information that is unique to the SM, and unknown and unpredictable to any person.

A Manufacturer SHALL have a documented process for SM initialisation and PUBKEY certification. The process SHALL be performed before an SM is delivered by the Manufacturer, and SHALL include at minimum:

- Complete the production of the SM, including loading firmware that has been produced by the STSA.
- In a physically secure facility and under dual control:
    - By means of physical inspection verify the integrity of all equipment to be used in this process.
    - Instruct the SM to generate a unique key pair (d_{SM}, Q_{SM}) and to return the public key Q_{SM}.
    - Use the Manufacturer's private key dMAN to certify the public key QSM, producing a $PUBKEY_{SM}$ certificate.
    - Ensure that Q_{SM} is protected against modification or substitution.
        - It SHALL NOT be possible for any individual to cause a chosen public key Q_{OP} to be signed under the Manufacturer's private key d_{MAN}.
        - By implication the generation and certificate of Q_{SM} must be tightly coupled.

9.2.1 Recommended process to generate and certify $PUBKEY_{SM}$

The following process is RECOMMENDED as a final step during manufacture of the SM:

- This process SHALL be performed under dual control.
- Perform a physical inspection of the SM to confirm that it is fully manufactured per specification and intact.
- Load into the SM firmware that has been approved by the STSA.
- Set the SM RTC to the current date and time (using a reliable clock).
- Instruct the SM to generate a unique $PUBKEY_{SM\text{-}NOSIG}$. The SM SHALL:
    - Set ***GNT*** (type TIMESTAMP) to the current date according to the RTC.
    - Generate a unique ECDH key pair (***d_{SM}***, ***Q_{SM}***) using **GENERATE-KEY()**.
        - ***d_{SM}*** is known only to the SM and SHALL NOT be revealed to any other party (including the SM manufacturer) under any circumstances.

- Set **ID_{SM} = BUILD-RECORD(**"SMID.1", ':', 4, MANUFACTURER, MID, GNT, **Fingerprint)** where **Fingerprint** is computed from **Q_{SM}** and other fields as described for PKID (section 7.1).
- Securely store **d_{SM}**, **Q_{SM}** and **ID_{SM}**. These values SHALL be stored within the cryptographic boundary in non-volatile memory that is erased on tamper, and SHALL include integrity protection (such as a checksum).
- Set **$PUBKEY_{SM\text{-}NOSIG}$ = BUILD-RECORD(**"PK.ECDH.1", '|', 5, **ID_{SM}**, **Q_{SM}HEX**, Expiry, ∅, ∅**)** where **Q_{SM}HEX** is encoded as described for PUBKEY (section 7.2). Expiry MAY be set to "99991231T115959Z".
- Return the unsigned **$PUBKEY_{SM\text{-}NOSIG}$**.

- Instruct the Manufacturer's HSM to sign the **$PUBKEY_{SM\text{-}NOSIG}$** to create a certified **$PUBKEY_{SM}$**.
  - The HSM SHALL require dual authentication of two trusted operators before performing the signature operation.
  - The HSM SHALL NOT create a signature if the GNT of **$PUBKEY_{SM\text{-}NOSIG}$** is greater than the Expiry of **$PUBKEY_{MAN}$**.
  - The HSM creates **$PUBKEY_{SM}$** (based on **$PUBKEY_{SM\text{-}NOSIG}$**), sets the Issuer to **$PKID_{MAN}$**, and generates the Signature using **d_{MAN}**.
- Store $PUBKEY_{SM}$, and discard $PUBKEY_{SM\text{-}NOSIG}$.

9.3 SM PUBKEY publication

Whenever the association between an SM and a public key is created or modified – such as after SM manufacture or maintenance (section 4.2) or a suspected key compromise – the SM Manufacturer SHALL publish the updated association to all KMCs:

- To revoke a public key with replacement follow the SM Initialisation and PUBKEY certification process to create an updated $PUBKEY_{SM}$.
- To revoke a public key without replacement construct and sign a $PUBKEY_{SM}$ with Q_{SM} = (0,0) (an invalid point).
- The Manufacturer adds each updated $PUBKEY_{SM}$ to a file-of-records (Appendix D – File-of-records format).
- The file sent to all KMCs (for example as an e-mail attachment).

10 KMC Initialisation

10.1 Prerequisites: KMC HSM

The KMC SHALL use a Hardware Security Module (HSM) to manage all keys and perform all cryptographic operations specified in this document.

- The HSM SHALL be certified to [FIPS PUB 140-2] Security Level 3 or higher, or to an equivalent evaluation level of a recognised standard for cryptographic modules such as [ISO 19790], or [PCI HSM].
- The STSA SHOULD maintain a Code of Practice detailing the security requirements for an HSM.

The HSM SHALL have:

- A high quality entropy source that has been assessed using statistical tests from NIST SP800-22. The HSM SHALL implement a continuous quality test on the output of the entropy source, for example by ensuring that adjacent blocks read from the source are distinct.
- A deterministic Random Bit Generator (RBG) seeded from the entropy source, with a security-strength of 192 bits or more.
    - The RBG SHALL comply with [ISO 18031], [NIST SP800-90], [ANSI X9.82] and/or [SEC 1].
- A real-time clock (RTC) for which the state is protected within the HSM's cryptographic boundary.
- Secure storage for sensitive data. All keys and sensitive data SHALL include integrity protection. Key separation and substitution prevention SHALL be assured (for example using techniques from [ISO 11568-2]). *See* Prerequisites: SM *(section 9.1) for permitted secure storage techniques.*
- Tested implementations of all cryptographic primitives (section 6) required by this specification.
- An authentic copy of the NIST P-384 domain parameters.

10.2 Prerequisites: KMC

The KMC SHALL have:

- KMCID (string of type IDENT), a unique name or identifier.
    - The STSA SHOULD provide a registry service for KMC names.
- SWID (string of type IDENT), a software application and version identifier.
- A list of Approved HWID values. The KMC SHALL NOT negotiate a KEK with (or transfer Vending Keys to) an SM unless that SM's HWID is in the Approved list.
- A list of Approved FWID values. The KMC SHALL NOT negotiate a KEK with (or transfer Vending Keys to) an SM unless that SM's FWID is in the Approved list.

The STSA SHOULD maintain a Code of Practice detailing the requirements for approving SM hardware and firmware (based on the SM Prerequisites in section 9.1).

The STSA SHOULD provide a registry service for Approved HWID and FWID values.

10.3 KMC Setup

Prior to accepting Vending Key Load Requests from SMs, the KMC SHALL:

- Generate an asymmetric digital signature key pair for the purpose of establishing KEKs with SMs.
    - The key pair SHALL be an ECC CDH key pair using the NIST P-384 domain parameters and having a security-strength of at least 192 bits, and SHALL be generated using an RBG having equivalent (or stronger) security-strength.
    - The key pair SHALL be generated and managed with respect to the principles of split knowledge and dual control.
    - The secret key SHALL be protected by an HSM.
- The key pair SHALL have a lifespan (Originator Usage Period) of at most 3 years (consistent with [NIST SP800-57 PART 1]).
- The public key SHALL be published as an unsigned PUBKEY record (referred to as $PUBKEY_{KMC}$; see section 7.2.) with record type “PK.ECDH.1”. The Expiry field SHALL reflect the end of the Originator Usage Period.
- A procedure to publish $PUBKEY_{KMC}$ SHALL be specified by KMC standards or operational documentation. Each recipient of the $PUBKEY_{KMC}$ SHALL check that the public key is not expired, and SHALL check the validity of the public key by manually confirming the public key’s fingerprint over an independent communication channel (either directly with the KMC or via a Trusted Third Party).
- When the KMC’s key pair expires, the KMC SHALL generate a new key pair and publish the public key in the manner prescribed by this section.

10.3.1 Recommended process to generate and publish $PUBKEY_{KMC}$

The following process is RECOMMENDED:

- The KMC selects a unique name KMCID (an IDENT).
- The KMC uses an HSM to generate and store a unique ECDSA key pair (d_{KMC}, Q_{KMC}) using the NIST P-384 domain parameters, in accordance with [ISO 14888-3], [FIPS PUB 186-3], [ANSI X9.62] and/or [SEC 1].
    - Generate a unique ECDH key pair (***d_{KMC}***, ***Q_{KMC}***) using **GENERATE-KEY()**.
        - ***d_{KMC}*** is known only to the KMC HSM and SHALL NOT be revealed to any other party under any circumstances.

- The KMC constructs a $PKID_{KMC}$ with *rectype* “KMC.1”, *Manufacturer* set to SWID, *MID* set to KMCID, and *GNT* the time at which d_{KMC} was generated.
    - Set ***GNT*** (type TIMESTAMP) to the current date according to the RTC.
    - Set ID_{KMC} = BUILD-RECORD(“KMCID.1”, ‘:’, 4, SWID, KMCID, GNT, Fingerprint) where Fingerprint is computed from QKMC and other fields as described for PKID (section 7.1).
- The KMC constructs a $PUBKEY_{KMC}$ with *rectype* “PK.ECDH.1” and Subject $PKID_{KMC}$. Expiry is at most 3 years after the *GNT*. The *Issuer* and *Signature* are empty.
    - Compute the **Expiry** date of the key pair ($\mathbf{d_{KMC}}$, $\mathbf{Q_{KMC}}$) as the **GNT** plus the Originator Usage Period (maximum 3 years).
    - Securely store $\mathbf{d_{KMC}}$, $\mathbf{Q_{KMC}}$, **Expiry** and $\mathbf{ID_{KMC}}$. These values SHALL be in secure storage and SHALL include integrity protection.
    - Set $\mathbf{PUBKEY_{KMC}}$ **= BUILD-RECORD(**“PK.ECDH.1”, ‘|’, 5, $\mathbf{ID_{KMC}}$, $\mathbf{Q_{KMC}HEX}$, **Expiry**, ∅, ∅**)** where $\mathbf{Q_{KMC}HEX}$ is encoded as described for PUBKEY (section 7.2).
- On demand by any SM Operator (vendors or meter manufacturer), the KMC sends to the SM Operator the $PUBKEY_{KMC}$ in record-in-email format (Appendix C – Record-in-email format).
- The SM Operator confirms the Fingerprint in the $PKID_{KMC}$ via a second channel (for example via the telephone or from the STSA website) then instructs their system to use the $PUBKEY_{KMC}$ in a Vending Key Load Request (section 11).

10.4 KMC operation

During operation the KMC will periodically receive updated information from SM Manufacturers and the STSA. Such updates SHALL be processed at the beginning of each day of operation, before processing Vending Key Load Requests. The integrity of the information SHALL be confirmed cryptographically or under dual control, and the information stored for future use.

10.4.1 SM Manufacturer $PUBKEY_{MAN}$ updates

When the KMC receives notification that an SM Manufacturer’s public key certificate $PUBKEY_{MAN}$ has been updated, the KMC should – with respect to the principle of dual control – verify the integrity and authenticity of the certificate then introduce it to the KMC HSM as a trusted certificate.

See the recommended process to generate and public $PUBKEY_{MAN}$ (section 8.1).

10.4.2 Approved HWID & FWID list updates

The KMC may receive from the STSA updated lists of Approved HWIDs and/or Approved FWIDs. The format and validation of these lists is beyond the scope of this specification.

10.4.3 Supply Group management instructions

The KMC may receive from Supply Group owners (or prospective owners) requests to register Supply Groups, update registration details, generate Vending Keys, or permit Vending Keys to be sent to specific SMs (identified by MANUFACTURER and MID). The format and validation of such requests is beyond the scope of this specification.

10.4.4 SM PUBKEY updates

The KMC will periodically receive files from SM Manufacturers containing updated SM public key certificates ($PUBKEY_{SM}$); see SM PUBKEY publication section 9.3.

The KMC SHALL validate each certificate using the KMC HSM and the Issuer's public key (an SM Manufacturer's $PUBKEY_{MAN}$ previously introduced as described in section 10.3). The KMC SHOULD check that each SM public key is unique. The certificate is stored in the KMC's database indexed by MANUFACTURER and MID, replacing any existing entry for the same SM; the new certificate's GNT (generation time) should be greater than that of the existing certificate.

11 SM Vending Key Load Request

When an SM requires vending keys from a KMC that SM SHALL perform the following process to create a Vending Key Load Request.

Input:

- $PUBKEY_{KMC}$

Software or SM firmware process (*error prefix "SM.1A" for software or "SM.1B" for firmware*):

- Parse $PUBKEY_{KMC}$ using PARSE-RECORD("PKECDH.1", '|', 5, $PUBKEY_{KMC}$), to retrieve ID_{KMC}, Q_{KMC}HEX and Expiry. Verify types of retrieved fields.

  If parsing fails then FAIL(*"SM.1A.1: Bad PUBKEY_KMC: failed to parse PUBKEY_KMC;"* ‖ *cause*).

  If Expiry is less than the current time then FAIL(*"SM.1A.2: Bad PUBKEY_KMC: certificate is expired"*).

SM firmware process (*error prefix "SM.1B"*):

- Input: ID_{KMC}, Q_{KMC}HEX.
- If this process has completed successfully within the last 60 seconds then FAIL(*"SM.1B.1: Load Request speed limit enforced; try again in 60 seconds").*
- Set Q_{KMC} = Octet-String-to-Point(BASE16-DECODE(Q_{KMC}HEX))

  On error FAIL(*"SM.1B.2: Bad PUBKEY_KMC: invalid representation for public key Q_KMC"*).

- Parse ID_{KMC} using PARSE-RECORD("KMCID.1", ':', 4, ID_{KMC}) to retrieve SWID, KMCID, $Serial_{KMC}$ and $Fingerprint_{KMC}$. Verify types of retrieved fields.

  If parsing fails then FAIL(*"SM.1B.3: Bad PUBKEY_KMC: failed to parse ID_KMC;"* ‖ *cause*).

  Verify the $Fingerprint_{KMC}$ using the retrieved fields and Q_{KMC} (see PKID, section 7.1), or FAIL(*"SM.1B.4: Bad PUBKEY_KMC: bad fingerprint in ID_KMC"*).

- Retrieve from secure storage the values d_{SM}, Q_{SM} and ID_{SM}, and check their integrity.

  If the integrity check fails then FAIL(*"SM.1B.5: Bad SM keys: stored key integrity failure"*).

- Parse ID_{SM} using PARSE-RECORD("SMID.1", ':', 4, ID_{SM}) to retrieve MANUFACTURER, MID, $Serial_{SM}$ and $Fingerprint_{SM}$. Verify types of retrieved fields.

  If parsing fails then FAIL(*"SM.1B.6: Bad SM keys: failed to parse ID_SM;"* ‖ *cause*).

  Verify the $Fingerprint_{SM}$ using the retrieved fields and Q_{SM} (see PKID, section 7.1), or FAIL(*"SM.1B.7: Bad SM keys: bad fingerprint in ID_SM"*).

- Retrieve NIST P-384 domain parameters and check their integrity.

  If the integrity check fails the FAIL(*"SM.1B.8: Bad SM keys: domain parameters corrupt"*).

- Use VALIDATE-KEY(Q_{KMC}) to provide assurance of validity of the KMC's public key.

  If validation fails then FAIL(*"SM.1B.9: Bad PUBKEY_KMC: public key Q_KMC failed full validation"*).

- Use VALIDATE-KEY(Q_{SM}) to provide assurance of validity of the SM's public key.

  If validation fails then FAIL(*"SM.1B.10: Bad SM keys: public key Q_SM failed full validation"*).

- Check that the SM has the correct value for its private key: using domain parameters (q, FR, a, b, G, n, h) = NIST P 384, check that d_{SM} is in the range [1, n-1] and if so compute $Q_{SM}' = d_{SM} G$ (scalar multiplication of a point on an elliptic curve).

  If d_{SM} is out of range or $Q_{SM}' \neq Q_{SM}$ then FAIL(*"SM.1B.11: Bad SM keys: SM private/public key mismatch"*).

- Set TVP_{KMC} to a TIMESTAMP the current time according to the SM's RTC.

  On error FAIL(*"SM.1B.12: Error creating VKLOADREQ: RTC fault"*).

- Generate an ephemeral key pair (d_E, Q_E) using GENERATE-KEY().

  Set $\mathbf{Q_ESTR}$ to Point-to-Octet-String(Q_E).

  On error FAIL(*"SM.1B.13: Error creating VKLOADREQ: ephemeral key generation fault").*

- Set Z_E = ECC-CDH(d_E, Q_{KMC}) then zeroise d_E.

  On error zeroise d_E and FAIL(*"SM.1B.14: Error creating VKLOADREQ: ephemeral CDH fault"*).

- Set Z_S = ECC-CDH(d_{SM}, Q_{KMC}).

  On error zeroise Z_E and FAIL(*"SM.1B.15: Error creating VKLOADREQ: static CDH fault"*).

- Set $\mathbf{Z} = Z_E \| Z_S$ then zeroise Z_E and Z_S.

- Construct **SharedInfo** = LVCONCAT("STS.KAA.1", ID_{SM}, ID_{KMC}, TVP_{KMC}).

- Set DKM = KDF-X963-SHA-384(**Z**, **SharedInfo**, 384) then zeroise Z.

  On error zeroise Z and FAIL(*"SM.1B.16: Error creating VKLOADREQ: KDF fault"*).

- Set $\mathbf{MacKey}_{192} \| \mathbf{KEK}_{192} = DKM_{384}$ then zeroise DKM. That is, take the leftmost 192 bits of DKM as MacKey, and the remaining 192 bits of DKM as KEK, then zeroise DKM.

- Construct $\mathbf{MacData_{SM}}$ = LVCONCAT("U_2", ID_{SM}, ID_{KMC}, Q_ESTR, TVP_{KMC}, HWID, FWID).

  Then compute $\mathbf{MacTag_{SM}}$ = HMAC-SHA-384-192(**MacKey**, $\mathbf{MacData_{SM}}$).

  On error zeroise MacKey and KEK, then FAIL(*"SM.1B.17: Error creating VKLOADREQ: MacTag_SM generation fault"*).

- Construct $\mathbf{MacData_{KMC}}$ = LVCONCAT("V2", ID_{KMC}, ID_{SM}, TVP_{KMC}, Q_ESTR).

Then compute **$ExpMacTag_{KMC}$** = HMAC-SHA-384-192(**MacKey**, **$MacData_{KMC}$**).

On error zeroise MacKey, KEK and $MacTag_{SM}$, then FAIL(*"SM.1B.18: Error creating VKLOADREQ: ExpMacTag_KMC generation fault"*).

- Set Q_EHEX (type 194H) = BASE16(Q_ESTR)

  Set $MacTag_{SM}$HEX (type 48H) = BASE16($MacTag_{SM}$)

- Construct the Vending Key Load Request:

  $VKLOADREQ_{SM}$ = BUILD-RECORD("VKLOAD.REQ.1", '|', 7, ID_{SM}, ID_{KMC}, TVP_{KMC}, HWID, FWID, Q_EHEX, $MacTag_{SM}$HEX). See also VKLOADREQ (section 7.3).

- Securely store KEK, $Fingerprint_{KMC}$, TVP_{KMC}, and $ExpMacTag_{KMC}$. The KEK SHALL be flagged with a 'pending' status that prevents it from being used by the HSM until a valid VKLOADRESP is received. Storage SHALL include integrity protection.

  $Fingerprint_{KMC}$, TVP_{KMC}, and $ExpMacTag_{KMC}$ will be used to verify the Key Load Response from the KMC.

- Output $VKLOADREQ_{SM}$.

Software or manual process:

- Log the Load Request to the software audit log (the log SHOULD NOT contain Q_EHEX, but SHOULD contain all other fields of $VKLOADREQ_{SM}$).
- Send the Vending Key Load Request $VKLOADREQ_{SM}$ to the KMC in record-in-email format (Appendix C – Record-in-email format).

12 KMC Vending Key Load Response

When a KMC receives a Vending Key Load Request (VKLOADREQ) from an SM, that KMC SHALL perform the following process to authenticate the SM, establish a shared KEK, and transfer Vending Keys to the SM.

Input:

- KMCID and ID_{KMC} (both known to the KMC)
- $VKLOADREQ_{SM}$

Software process (*error prefix "KMC.2A"*):

- Log the Load Request to the KMC audit log (the log SHOULD NOT contain Q_EHEX, but SHOULD contain all other fields of $VKLOADREQ_{SM}$).
- Parse $VKLOADREQ_{SM}$ using PARSE-RECORD("VKLOAD.REQ.1", '|', 7, $VKLOADREQ_{SM}$) to retrieve REQ_ID_{SM}, REQ_ID_{KMC}, TVP_{KMC}, HWID, FWID, Q_EHEX, and $MacTag_{SM}$HEX. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2A.1: Bad VKLOADREQ: failed to parse VKLOADREQ_SM;"* ‖ *cause*).
- Parse REQ_ID_{KMC} using PARSE-RECORD("KMCID.1", ':', 4, REQ_ID_{KMC}) to retrieve REQ_KMCID. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2A.2: Bad VKLOADREQ: failed to parse ID_KMC;"* ‖ *cause*).

  The Fingerprint of REQ_ID_{KMC} is verified later by comparison against a known ID_{KMC}.
- If REQ_KMCID ≠ KMCID then FAIL(*"KMC.2A.3: Bad VKLOADREQ: key load request sent to wrong KMC"*).
- If REQ_ID_{KMC} ≠ ID_{KMC} then FAIL(*"KMC.2A.4: Bad VKLOADREQ: key load request used old PUBKEY_KMC"*).
- Parse REQ_ID_{SM} using PARSE-RECORD("SMID.1", ':', 4, ID_{SM}) to retrieve MANUFACTURER, MID, and GNT. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2A.5: Bad VKLOADREQ: failed to parse ID_SM;"* ‖ *cause*).

  The Fingerprint of REQ_ID_{SM} is verified later by comparison against a known ID_{SM}.
- Find in the KMC database the $PUBKEY_{SM}$ and $LastTVP_{KMC}$ associated with MANUFACTURER and MID. This $PUBKEY_{SM}$ was securely distributed to the KMC by the SM Manufacturer.

  If no matching PUBKEY is found then FAIL(*"KMC.2A.6: KMC data out of date: no PUBKEY_SM found for SM; KMC may need update file from SM manufacturer"*).
- Parse $PUBKEY_{SM}$ using PARSE-RECORD("PKECDH.1", '|', 5, $PUBKEY_{SM}$), to retrieve ID_{SM} and Issuer. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2A.7: Error in KMC data: failed to parse PUBKEY_SM;"* ‖ *cause*).

- If $REQ\_ID_{SM} \neq ID_{SM}$ then FAIL(*"KMC.2A.8: KMC data out of date: mismatch between requesting ID_SM and database; KMC may have old PUBKEY_SM"*).
- Find in the KMC database the trusted $PUBKEY_{MAN}$ associated with the issuer. This $PUBKEY_{MAN}$ was distributed to the KMC by the SM Manufacturer and introduced to the KMC HSM under dual control (see sections 8 and 8.1).

  If no matching $PUBKEY_{MAN}$ is found then FAIL(*"KMC.2A.9: Error in KMC data: unknown Issuer for PUBKEY_SM; cannot validate certificate"*).
- If $TVP_{KMC} \leq LastTVP_{KMC}$ then FAIL(*"KMC.2A.10: Bad VKLOADREQ: old timestamp (TVP) in VKLOADREQ_SM; possible out-of-order request or replay"*).
- The KMC SHOULD check that TVP_{KMC} is within an acceptable window around the current time (according to the system clock). The window SHOULD be software configurable. A window of (now – 30 days) to (now + 3 days) is RECOMMENDED.

  If TVP_{KMC} is outside the acceptable window then FAIL(*"KMC.2A.11: Bad VKLOADREQ: timestamp (TVP) outside acceptable window; possible delayed or future-dated request"*).
- If HWID is not in the list of Approved HWIDs then FAIL*("KMC.2A.12: Bad VKLOADREQ: SM hardware model not approved"*).
- If FWID is not in the list of Approved FWIDs then FAIL*("KMC.2A.13: Bad VKLOADREQ: SM firmware not approved"*).

KMC HSM firmware process (*error prefix "KMC.2B"*):

- Input: $PUBKEY_{MAN}$, $PUBKEY_{SM}$, ID_{KMC}, TVP_{KMC}, HWID, FWID, Q_EHEX, $MacTag_{SM}HEX$.
- Check types of inputs TVP_{KMC} (TIMESTAMP), HWID (IDENT), FWID (IDENT), Q_EHEX (194H), and $MacTag_{SM}HEX$ (48H).

  If type checking fails then FAIL(*"KMC.2B.1: Bad VKLOADREQ: bad encoding in input;"* ‖ *cause*).
- Set Q_ESTR = BASE16-DECODE(Q_EHEX).

  Set Q_E = Octet-String-to-Point(Q_ESTR).

  If conversion fails then FAIL(*"KMC.2B.2: Bad VKLOADREQ: bad representation for Q_E"*).
- Verify that $PUBKEY_{MAN}$ is a trusted certificate or FAIL(*"KMC.2B.3: Error in KMC data: certificate PUBKEY_MAN is not trusted"*).
- Parse $PUBKEY_{MAN}$ using PARSE-RECORD("PKECDSA.1", '|', 5, $PUBKEY_{MAN}$), to retrieve ID_{MAN}, $Q_{MAN}HEX$ and $Expiry_{MAN}$. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2B.4: Error in KMC data: failed to parse PUBKEY_MAN;"* ‖ *cause*).

  Set Q_{MAN} = Octet-String-to-Point(BASE16-DECODE($Q_{MAN}HEX$)).

  If conversion fails then FAIL(*"KMC.2B.5: Error in KMC data: bad representation for Q_MAN"*).

Parse ID_{MAN} using PARSE-RECORD("SMMAN.1", ':', 4, ID_{MAN}) to retrieve Fingerprint and other fields. Verify types of retrieved fields.

If parsing fails then FAIL(*"KMC.2B.6: Error in KMC data: failed to parse ID_MAN;"* ‖ *cause*).

Verify the Fingerprint using the retrieved fields and Q_{MAN} (see PKID, section 7.1), or FAIL(*"KMC.2B.7: Error in KMC data: bad fingerprint in ID_MAN"*).

- Parse $PUBKEY_{SM}$ using PARSE-RECORD("PKECDH.1", '|', 5, $PUBKEY_{SM}$), to retrieve ID_{SM}, Q_{SM}HEX, $Expiry_{SM}$, Issuer and Signature. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2B.8: Error in KMC data: failed to parse PUBKEY_SM;"* ‖ *cause*).

  Set Q_{SM} = Octet-String-to-Point(BASE16-DECODE(Q_{SM}HEX)).

  If conversion fails then FAIL(*"KMC.2B.9: Error in KMC data: bad representation for Q_SM"*).

  Parse ID_{SM} using PARSE-RECORD("SMID.1", ':', 4, ID_{SM}) to retrieve $Serial_{SM}$, Fingerprint and other fields. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2B.10: Error in KMC data: failed to parse ID_SM;"* ‖ *cause*).

  Verify the Fingerprint using the retrieved fields and Q_{SM} (see PKID, section 7.1), or FAIL(*"KMC.2B.11: Error in KMC data: bad fingerprint in ID_SM"*).

- If Issuer ≠ ID_{MAN} then FAIL(*"KMC.2B.12: Error verifying VKLOADREQ: cannot verify SM certificate; wrong Issuer key presented"*).

- If $Serial_{SM}$ > $Expiry_{MAN}$ then FAIL(*"KMC.2B.13: Error verifying VKLOADREQ: invalid SM certificate; GNT postdates Issuer expiry"*).

- If $Expiry_{SM}$ is less than the current time (from the HSM RTC) then FAIL(*"KMC.2B.14: Error verifying VKLOADREQ: SM certificate is expired"*).

- Retrieve from secure storage the values d_{KMC}, Q_{KMC}, $Expiry_{KMC}$ and ID_{KMC}, and check their integrity.

  If the integrity check fails then FAIL(*"KMC.2B.15: Bad KMC keys: stored key integrity failure"*).

- If $Expiry_{KMC}$ is less than the current time (from the HSM RTC) then FAIL(*"KMC.2B.16: Bad KMC keys: PUBKEY_KMC has expired"*).

- Parse ID_{KMC} using PARSE-RECORD("KMCID.1", ':', 4, ID_{KMC}) to retrieve SWID, KMCID, GNT and Fingerprint. Verify types of retrieved fields.

  If parsing fails then FAIL(*"KMC.2B.17: Bad KMC keys: failed to parse ID_KMC;"* ‖ *cause*).

  Verify the Fingerprint using the retrieved fields and Q_{KMC} (see PKID, section 7.1), or FAIL(*"KMC.2B.18: Bad KMC keys: bad fingerprint in ID_KMC"*).

- Retrieve NIST P-384 domain parameters and check their integrity.

If the integrity check fails the FAIL(*"KMC.2B.19: Bad KMC keys: domain parameters corrupt"*).

- Use VALIDATE-KEY(Q_{MAN}) to provide assurance of validity of the SM Manufacturer's public key.

  If validation fails then FAIL(*"KMC.2B.20: Error in KMC data: public key Q_MAN failed full validation"*).

- Verify Signature of $PUBKEY_{SM}$ using Q_{MAN} as described in section 7.2.

  If Signature is invalid then FAIL(*"KMC.2B.21: Error verifying VKLOADREQ: invalid signature on SM certificate"*).

- Use VALIDATE-KEY(Q_{SM}) to provide assurance of validity of the SM's public key.

  If validation fails then FAIL(*"KMC.2B.22: Error in KMC data: public key Q_SM failed full validation"*).

- Use VALIDATE-KEY(Q_{KMC}) to provide assurance of validity of the KMC's public key.

  If validation fails then FAIL(*"KMC.2B.23: Bad KMC keys: public key Q_KMC failed full validation"*).

- Check that the KMC has the correct value for its private key: using domain parameters (q, FR, a, b, G, n, h) = NIST P-384, check that d_{KMC} is in the range [1, n-1] and if so compute $Q_{KMC}' = d_{KMC}$ G (scalar multiplication of a point on an elliptic curve).

  If d_{KMC} is out of range or $Q_{KMC}' \neq Q_{KMC}$ then FAIL(*"KMC.2B.24: Bad KMC keys: KMC private/public key mismatch"*).

- Use VALIDATE-KEY(Q_E) to provide assurance of validity of the SM's ephemeral public key.

  If validation fails then FAIL(*"KMC.2B.25: Error verifying VKLOADREQ: public key Q_E failed full validation"*).

- Set Z_E = ECC-CDH(d_{KMC}, Q_E).

  On error FAIL(*"KMC.2B.26: Error verifying VKLOADREQ: ephemeral CDH fault"*).

- Set Z_S = ECC-CDH(d_{KMC}, Q_{SM}).

  On error zeroise Z_E and FAIL(*"KMC.2B.27: Error verifying VKLOADREQ: static CDH fault"*).

- Set **Z** = $Z_E \| Z_S$ then zeroise Z_E and Z_S.

- Construct **SharedInfo** = LVCONCAT("STS.KAA.1", ID_{SM}, ID_{KMC}, TVP_{KMC}).

- Set DKM = KDF-X963-SHA-384(**Z**, **SharedInfo**, 384) then zeroise Z.

  On error zeroise Z and FAIL(*"KMC.2B.28: Error verifying VKLOADREQ: KDF fault"*).

- Set $\textbf{MacKey}_{192} \| \textbf{KEK}_{192} = DKM_{384}$ then zeroise DKM. That is, take the leftmost 192 bits of DKM as MacKey, and the remaining 192 bits of DKM as KEK, then zeroise DKM.

- Construct **$MacData_{SM}$** = LVCONCAT("U_2", ID_{SM}, ID_{KMC}, Q_ESTR, TVP_{KMC}, HWID, FWID).

  Then compute **$ExpMacTag_{SM}$** = HMAC-SHA-384-192(**MacKey**, **$MacData_{SM}$**).

  On error zeroise MacKey and KEK, then FAIL(*"KMC.2B.29: Error verifying VKLOADREQ: ExpMacTag_SM generation fault"*).

- If $MacTag_{SM}$HEX ≠ BASE16($ExpMacTag_{SM}$) then zeroise MacKey and KEK, and FAIL(*"KMC.2B.30: Bad VKLOADREQ: bad key confirmation from SM"*).

- Construct **$MacData_{KMC}$** = LVCONCAT("V2", ID_{KMC}, ID_{SM}, TVP_{KMC}, Q_ESTR).

  Then compute **$MacTag_{KMC}$** = HMAC-SHA-384-192(**MacKey**, **$MacData_{KMC}$**).

  On error zeroise MacKey, KEK and $MacTag_{SM}$, then FAIL(*"KMC.2B.31: Error creating VKLOADRESP: MacTag_KMC generation fault"*).

- Set $MacTag_{KMC}$HEX (type 48H) = BASE16($MacTag_{KMC}$)

- Construct the Vending Key Load Response:

  $VKLOADRESP_{KMC}$ = BUILD-RECORD("VKLOAD.RESP.1", '|', 4, ID_{KMC}, ID_{SM}, TVP_{KMC}, $MacTag_{KMC}$HEX).

- Securely store KEK.
  KEK will be used to wrap Vending Keys for transfer to the SM.

- Output $VKLOADRESP_{KMC}$.

Mixed software and SM firmware process (*error prefix "KMC.2C" for software or "KMC.2D" for firmware*):

- Store TVP_{KMC} as $LastTVP_{KMC}$ associated with SM MANUFACTURER and MID, or FAIL(*"KMC.2C.1: Error creating VKLOADRESP: LAST_TVP_KMC storage error"* ‖ *cause*).

- Create a **Key Load File** as a file-of-records (Appendix D – File-of-records format), and add the $VKLOADRESP_{KMC}$ as the first record.

- Find all Vending Keys authorised for use with the SM (by MANUFACTURER and MID).

  For each authorised vending key ***VK***:

  - Use the KMC HSM to build a WRAPPED-KEY record (section 7.5) – protected by the KEK – for the ***VK*** and associated attributes.
    - The maximum permitted size of ***VK*** is 160 bits.
    - For a given KEK, the KMC HSM SHALL ensure that all WRAPPED-KEY records have distinct Nonces.
    - Errors raised during this process SHOULD use the error prefix *"KMC.2D"*.
  - Append the WRAPPED-KEY to the **Key Load File**.
  - Update the KMC database and audit log to reflect the distribution of the **VK** to the SM (identified by MANUFACTURER and MID).

- Log the Load Response to the KMC audit log (the log SHOULD contain all fields of $VKLOADRESP_{KMC}$).
- Send the **Key Load File** to the SM (for example as an e-mail attachment).

13 SM KEK Confirmation and Vending Key Import

When an SM receives a Vending Key Load Response (VKLOADRESP) from a KMC, that SM SHALL perform the following process to authenticate the KMC, confirm the shared KEK, and import the Vending Keys to the SM.

Input:

- **Key Load File** (file-of-records) containing $VKLOADRESP_{KMC}$ and zero or more WRAPPED-KEY records.

Mixed software and SM firmware process (*error prefix "SM.3A" for software or "SM.3B" for firmware*):

- Parse **Key Load File** to recover $VKLOADRESP_{KMC}$ and WRAPPED-KEY records.

  If parsing fails or if the file checksum is incorrect then FAIL(*"SM.3A: Bad Key Load File;"* ‖ *cause*).

- The SM SHALL perform the following process to finish establishing the KEK:
  - Input: $VKLOADRESP_{KMC}$.
  - Retrieve from secure storage the values KEK, $Fingerprint_{KMC}$, TVP_{KMC}, and $ExpMacTag_{KMC}$ (all stored while generating the $VKLOADREQ_{SM}$, section 12) and check their integrity.

    If the integrity check fails then FAIL(*"SM.3B.1: Error verifying VKLOADRESP: key agreement session integrity failure"*).
  - Set NOW to the current time according to the SM's RTC.
  - If TVP_{KMC} < (NOW – 60 days) then FAIL(*"SM.3B.2: Error verifying VKLOADRESP: key agreement session timeout"*).
  - Parse $VKLOADRESP_{KMC}$ using PARSE-RECORD("VKLOAD.RESP.1", '|', 4, $VKLOADRESP_{KMC}$), to retrieve $RESP\_ID_{KMC}$, $RESP\_ID_{SM}$, $RESP\_TVP_{KMC}$ and $MacTag_{KMC}$HEX. Verify types of retrieved fields.

    If parsing fails then FAIL(*"SM.3B.3: Bad VKLOADRESP: failed to parse VKLOADRESP_KMC;"* ‖ *cause*).
  - Parse ID_{KMC} using PARSE-RECORD("KMCID.1", ':', 4, ID_{KMC}) to retrieve $RESP\_Fingerprint_{KMC}$. Verify types of retrieved fields.

    If parsing fails then FAIL(*"SM.3B.4: Bad VKLOADRESP: failed to parse ID_KMC;"* ‖ *cause*).
  - Retrieve from secure storage the ID_{SM}, and check its integrity.

    If the integrity check fails then FAIL(*"SM.3B.5: Bad SM keys: stored key integrity failure"*).
  - If $RESP\_ID_{SM} \neq ID_{SM}$ then FAIL(*"SM.3B.6: Destination error: VKLOADRESP_KMC is for a different SM"*).
  - If $RESP\_Fingerprint_{KMC} \neq Fingerprint_{KMC}$ then FAIL(*"SM.3B.7: Destination error: VKLOADRESP_KMC is for a different key agreement session (with a different KMC)"*).

This partial check on ID_{KMC} is not required for protocol security – its presence helps to identify and correct some Vending Key Load Response management errors.

- If $RESP\_TVP_{KMC} \neq TVP_{KMC}$ then FAIL(*"SM.3B.8: Bad VKLOADRESP: wrong timestamp (TVP) in VKLOADRESP_KMC; possible expired or out-of-order response"*).
- If $MacTag_{KMC}HEX \neq BASE16(ExpMacTag_{KMC})$ then FAIL(*"SM.3B.9: Bad VKLOADRESP: bad key confirmation from KMC"*).
- Zeroise TVP_{KMC} and $ExpMacTag_{KMC}$ from secure storage.

  Update the KEK status flag to indicate that the KEK may be used.

  ID_{KMC} or elements thereof are not security sensitive and may be retained.
- Output a success indicator.

- For each Vending Key required by the SM Operator, the protected Vending Key (a WRAPPED-KEY record) SHALL be imported into the SM:
  - The WRAPPED-KEY record may be parsed by software or by the SM.
  - The SM SHALL use **AES-192-CCM$_{DEC}$(KEK,** Nonce**,** Attributes**,** ProtectedKey**)** to verify the integrity of the Vending Key and Attributes and to decrypt the Vending Key (maximum size 160 bits).
  - The SM SHALL protect the cleartext value of the Vending Key and SHALL ensure that this value is not exposed outside the cryptographic boundary under any circumstances.
  - The SM SHALL protect the association between the Vending Key and its Attributes, and SHALL ensure that Attributes and not substituted or modified.
  - The SM SHALL securely store the Vending Key and associated Attributes. *See Prerequisites: SM (section 9.1) for permitted secure storage techniques.*
- Once all required Vending Keys have been imported the SM Operator or operating software SHALL instruct the SM that key transfer is complete:
  - The SM SHALL zeroise the KEK, preventing further WRAPPED-KEYs from being imported.
  - The SM MAY retain ID_{KMC} or elements thereof to assist in Vending Key management.

14 End-of-life and key compromise procedures

When any participating entity in the STS Key Management infrastructure reaches end-of-life, or the secret key material of that entity is compromised or suspected to be compromised, certain actions must be taken to ensure the integrity of the Key Management System.

This section details the essential aspects of end-of-life and key compromise procedures for various entities. All SM Manufacturers and KMCs SHALL have documented procedures for handling end-of-life and key compromise. Such procedures SHALL include at minimum the relevant actions specified in this section. KMCs SHALL require SM Operators to follow documented SM procedures as a condition of service.

14.1 SM Manufacturer

14.1.1 End-of-life

- The SM Manufacturer SHALL destroy its private ECDSA key d_{MAN}.
- The Manufacturer SHALL notify the STSA and all KMCs that it will not be producing further SMs.
- KMCs SHALL NOT accept further $PUBKEY_{SM}$ updates from the SM Manufacturer.
- There is no need to revoke the Manufacturer's public key certificate $PUBKEY_{MAN}$ – existing SMs can continue to establish KEKs with the KMC until they reach end-of-life or the Manufacturer's private ECDSA key d_{MAN} is compromised.

14.1.2 Storage Master Key (SMK) or private ECDSA key (d_{MAN}) compromise

- The SM Manufacturer MAY create a self-signed key revocation certificate using its private ECDSA key d_{MAN}. *The details of such a revocation certificate are beyond the scope of this document.*
- The Manufacturer SHALL destroy its private ECDSA key d_{MAN}.
- The Manufacturer SHALL notify the STSA of the (suspected) key compromise.
- The Manufacturer SHALL notify all KMCs that they can no longer trust the certificate $PUBKEY_{MAN}$. KMCs SHALL revoke trust in the certificate.
- The Manufacturer SHALL follow the Manufacturer Setup process (section 8) to generate and distribute a new $PUBKEY_{MAN\text{-}NEW}$.
- The Manufacturer SHALL investigate the integrity of its database of $PUBKEY_{SM}$ certificates and SHALL publish new certificates (signed by the Manufacturer's new private key) for those $PUBKEY_{SM}$ records found to be trustworthy (that is, existing certificates will be re-signed).

14.2 Security Module

14.2.1 End-of-life

- The SM Operator SHALL follow the documented Manufacturer procedure to destroy all secret data in the SM.

- The SM Operator SHALL notify the SM Manufacturer (possibly via a KMC) that the SM has been decommissioned.
- The SM Manufacturer SHALL publish a suitable revocation certificate to all KMCs. See SM PUBKEY publication (section 9.3).

14.2.2 Private ECC CDH key (d_{SM}) compromise

- The SM Operator SHALL follow the documented Manufacturer procedure to destroy all secret data in the SM.
- The Operator SHALL decommission the SM (see 14.2.1) or return the SM to the Manufacturer for maintenance.
- The Operator SHALL notify all KMCs of the compromise, and SHALL include logs of legitimate Vending Key Load Requests sent by the SM (at minimum logs of all Load Requests since the suspected date of compromise).
- All affected KMCs SHALL review their audit logs in conjunction with the logs of legitimate Load Requests supplied by the SM Operator to determine if unauthorised Vending Key Load Requests have been processed.
    - If unauthorised requests have been processed then the KMC(s) SHALL identify the affected Vending Keys and treat them as compromised (see 0).
    - If no unauthorised requests have been processed then no further action is required. *The secrecy of Vending Keys is still protected by the SM's cryptographic boundary and the forward secrecy property of the KEK agreement protocol.*

14.2.3 Storage Master Key (SMK) or Vending Key (VK) compromise

- The SM Operator SHALL determine the originating KMCs of the compromised VK(s).
- The SM Operator SHALL notify the STSA and originating KMCs of the compromise.
- KMCs SHALL proceed according to their procedures for VK compromise (section 14.3.2).

14.3 Key Management Centre

14.3.1 End-of-life

- The KMC SHALL notify the STSA and all SM Manufacturers that it will be ceasing operation.
- The KMC SHALL send a notice to all SM Operators that use its services. Operators SHOULD NOT make further use of the KMC's certificate $PUBKEY_{KMC}$.
- The KMC SHALL send a notice to all SG Owners that rely on its services. It will be necessary to migrate Supply Group keys and data to another KMC. *The details of such a migration are beyond the scope of this document.*

- The KMC SHALL destroy its private ECC CDH key d_{KMC}, its Storage Master Key (SMK), all key components and keys backups, and all data backups.

14.3.2 Key compromise

The compromise or suspected compromise of keys protected by or used by the KMC has a broad impact on the STS Key Management infrastructure, and is beyond the scope of this document.

KMC standards SHALL specify procedures or procedural requirements to handle the event in which any of the following keys are compromised or suspected to be compromised:

- The KMC's ECC CDH private key (d_{KMC});
- The KMC's Storage Master Key (SMK) or any component thereof;
- One or more Vending Keys (VKs) for Supply Groups served by the KMC.

Such procedures SHALL include notification of the STSA, affected SG Owners, and affected SM Operators.

15 Bibliography

[ANSI X9.62] ANS X9.62-2005 Public Key Cryptography for the Financial Services Industry – The Elliptic Curve Digital Signature Algorithm (ECDSA)

[ANSI X9.63] X9.63-2001 Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography

[ANSI X9.82] ANSI X9.82-1:2006 Random Number Generation – Part 3: Deterministic Random Bit Generator Mechanisms

[ANSI X9.102] ANSI X9.102:2008 Symmetric Key Cryptography For the Financial Services Industry – Wrapping of Keys and Associated Data, June 2008.

[CM10] Chen & Mitchell, “Parsing ambiguities in authentication and key establishment protocols”, 2010

[CRC-CAT] CRC RevEng: Catalogue of parametrised CRC algorithms (MODBUS)
HTTP://REVENG.SOURCEFORGE.NET/CRC-CATALOGUE/16.HTM#CRC.CAT.MODBUS

[FIPS PUB 140-2] Security Requirements for Cryptographic Modules, May 2001
HTTP://CSRC.NIST.GOV/PUBLICATIONS/FIPS/FIPS140-2/FIPS1402.PDF

[FIPS PUB 180-4] Secure Hash Standard (SHS), March 2012
HTTP://CSRC.NIST.GOV/PUBLICATIONS/FIPS/FIPS180-4/FIPS-180-4.PDF

[FIPS PUB 197] Advanced Encryption Standard (AES), November 2001
HTTP://CSRC.NIST.GOV/PUBLICATIONS/FIPS/FIPS197/FIPS-197.PDF

[FIPS PUB 198-1] The Keyed-Hash Message Authentication Code (HMAC), July 2008
HTTP://CSRC.NIST.GOV/PUBLICATIONS/FIPS/FIPS198-1/FIPS-198-1_FINAL.PDF

[ISO 9798-4] ISO/IEC 9798-4:1999 Information technology – Security techniques – Entity authentication – Part 4: Mechanisms using a cryptographic check function

[ISO 10116] ISO/IEC 10116:2006 Information technology – Security techniques – Modes of operation for an n-bit block cipher

[ISO 11568-2] ISO 11568-2:2005 Banking – Key management (retail) – Part 2: Symmetric ciphers, their key management and life cycle

[ISO 11770-2] ISO/IEC 11770-2:2007 Information technology – Security techniques – Key management – Part 2: Mechanisms using symmetric techniques

[ISO 13569]	ISO/TR 13569:2005 Financial services – Information security guidelines
[ISO/TR 14742]	ISO/TR 14742:2010 Financial services – Recommendations on cryptographic algorithms and their use, July 2010
[ISO 18031]	ISO/IEC 18031:2011 Information technology – Security techniques – Random bit generation
[ISO 19790]	ISO/IEC 19790:2012 Information technology – Security techniques – Security requirements for cryptographic modules
[ITU X.680]	Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation, July 2002 HTTP://WWW.ITU.INT/ITU-T/STUDYGROUPS/COM17/LANGUAGES/X.680-0207.PDF
[Lammert]	On-line CRC calculation and free library HTTP://WWW.LAMMERTBIES.NL/COMM/INFO/CRC-CALCULATION.HTML
[NIST SP800-38C]	NIST Special Publication 800-38C Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, July 2007 HTTP://CSRC.NIST.GOV/PUBLICATIONS/NISTPUBS/800-38C/SP800-38C_UPDATED-JULY20_2007.PDF
[NIST SP800-57 Part 1]	NIST Special Publication 800-57 Recommendation for Key Management – Part 1: General (Revised), March 2007
[NIST SP800-90]	NIST Special Publication 800-90 Recommendation for Random Number Generation Using Deterministic Random Bit Generators, January 2012 HTTP://CSRC.NIST.GOV/PUBLICATIONS/NISTPUBS/800-90A/SP800-90A.PDF
[NIST SP800-131A]	NIST Special Publication 800-131A Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, January 2011
[NIST SP800-152 DRAFT]	Requirements and Desirable Features of U.S. Federal Cryptographic Key Management Systems, DRAFT August 2012
[NISTIR 7628]	Guidelines for Smart Grid Cyber Security: Vol. 1, Smart Grid Cyber Security Strategy, Architecture, and High-Level Requirements, August 2010
[PCI HSM]	Payment Card Industry (PCI) PIN Transaction Security (PTS) Hardware Security Module (HSM) Security Requirements, Version 2.0, May 2012
[POSIX RE]	Wikipedia: Regular expression (POSIX) HTTP://EN.WIKIPEDIA.ORG/WIKI/REGULAR_EXPRESSION#POSIX

[RFC 2104]	HMAC: Keyed-Hashing for Message Authentication, February 1997 HTTP://WWW.IETF.ORG/RFC/RFC2104.TXT
[RFC 2119]	Key words for use in RFCs to Indicate Requirement Levels, March 1997 HTTP://WWW.IETF.ORG/RFC/RFC2119.TXT
[RFC 2144]	The CAST-128 Encryption Algorithm, May 1997 HTTP://WWW.IETF.ORG/RFC/RFC2144.TXT
[RFC 2994]	A Description of the MISTY1 Encryption Algorithm, November 2000 HTTP://TOOLS.IETF.ORG/HTML/RFC2994
[RFC 3610]	Counter with CBC-MAC (CCM), September 2003 HTTP://WWW.IETF.ORG/RFC/RFC3610.TXT
[RFC 4648]	The Base16, Base32, and Base64 Data Encodings, October 2006 HTTP://TOOLS.IETF.ORG/HTML/RFC4648#SECTION-8
[RFC 5869]	HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010 HTTP://TOOLS.IETF.ORG/HTML/RFC5869
[ROCKSOFT]	A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS, August 1993 HTTP://WWW.ROSS.NET/CRC/DOWNLOAD/CRC_V3.TXT
[SANS 1524-6-10]	SANS 1524-6-10:2010 Electricity payment systems – Part 6-10: Interface standards – Online vending server – Vending clients
[SEC 1]	SEC1: Elliptic Curve Cryptography version 2.0, May 2009 HTTP://WWW.SECG.ORG/DOWNLOAD/AID-780/SEC1-V2.PDF
[SEC 2]	SEC 2: Recommended Elliptic Curve Domain Parameters version 2.0, January 2010 HTTP://WWW.SECG.ORG/DOWNLOAD/AID-784/SEC2-V2.PDF
[STS COP 402-1]	STS COP 402-1:2011 Standard Transfer Specification (STS) – CODE OF PRACTICE FOR THE MANAGEMENT OF TOKEN ID ROLLOVER
[W:ASC]	Wikipedia: ASCII HTTP://EN.WIKIPEDIA.ORG/WIKI/ASCII
[W:BCD]	Wikipedia: Binary-coded decimal HTTP://EN.WIKIPEDIA.ORG/WIKI/BINARY-CODED_DECIMAL
[W:CERT]	Wikipedia: Public key certificate HTTP://EN.WIKIPEDIA.ORG/WIKI/PUBLIC_KEY_CERTIFICATE
[W:EBNF]	Wikipedia: Extended Backus-Naur Form HTTP://EN.WIKIPEDIA.ORG/WIKI/EXTENDED_BACKUS%E2%80%93NAUR_FORM

[W:END]	Wikipedia: Endianness HTTP://EN.WIKIPEDIA.ORG/WIKI/ENDIANNESS
[W:HEX]	Wikipedia: Hexadecimal HTTP://EN.WIKIPEDIA.ORG/WIKI/HEXADECIMAL
[W:LEX]	Wikipedia: Lexicographical order HTTP://EN.WIKIPEDIA.ORG/WIKI/LEXICOGRAPHICAL_ORDER
[W:OCT]	Wikipedia: Octet HTTP://EN.WIKIPEDIA.ORG/WIKI/OCTET_(COMPUTING)
[W:SHA-1]	Wikipedia: SHA-1 HTTP://EN.WIKIPEDIA.ORG/WIKI/SHA-1

16 Appendix A – example VKLOADRESP (informative)

The VKLOADRESP consists of a single record type record (type VKLOAD.RESP.1) followed by one or more key records (type KEY.1), and terminated with a HMAC of the entire record (#2FC…..).

A sample VKLOADRESP is shown below (5 keys included). Note that all records are fully described in the relevant sections of this specification.

VKLOAD.RESP.1|KMCID.1:Prism:K0001:20160418T121717Z:52204DE9EEFA6EB8:E7BF|SMID.1:Prism:94000507:20160506T095338Z:F184871DC4F23CB0:1F8C|20160506T122741Z|C9A5161F864E1978435A2CEAA611930F37824EDCE6252CEC|71CC

KEY.1|A8556C52BA3345996C1551DC|ACT20160425T114321Z;BDT19930101T000000Z;CLM44fa0000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCV17D819;KEN255;KRN1;KTC2;SBMFFFF;SGC0000112233;SGNSURELOAD KMC TEST VUDK 1;ULM1000;|849B9665C2BFFB75BF58629A0A057D528888559693DB293A|8F06

KEY.1|E1F557BEE6442E0FC5747E6B|ACT20160504T220000Z;BDT19930101T000000Z;CLM44fa0000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCV707844;KEN255;KRN2;KTC1;SBMFFFF;SGC0000112233;SGNSURELOAD KMC TEST VUDK
1;ULM1000;|C4967CD420827099DE6A1E5E670BA559AE6A3BC034BA090E|5D2D

KEY.1|49C0495A0A6C26351C44A0CB|ACT20160430T220000Z;BDT19930101T000000Z;CLM44fa0000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCV3A3273;KEN255;KRN3;KTC1;SBMFFFF;SGC0000112233;SGNSURELOAD KMC TEST VUDK 1;ULM1000;|DC29A26BBB03D794649E45856B8031B57323A090341CB747|576D

KEY.1|5EE5F985A1186F8A5DD1175F|ACT20140813T000000Z;BDT19930101T000000Z;CLM461c4000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCVC33F45;KEN255;KRN1;KTC2;SBM0001;SGC0000123456;SGNACME;ULM10000;|7865DC2B97755CFFA8B5A83C34D1AB8EB6955666F5C78A0B|25EB

KEY.1|6EE438A953F883942F60DDD8|ACT20160502T220000Z;BDT19930101T000000Z;CLM461c4000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCV55E354;KEN255;KRN2;KTC1;SBM0001;SGC0000123456;SGNACME;ULM10000;|9918600913859398D044C8D1E86A10EA5ECBF675E861862D|19FA

#2FC215350D8A718CD22783F5D81F8E33F6B46337

16.1.1 Record Type VKLOAD.RESP.1

VKLOAD.RESP.1|KMCID.1:Prism:K0001:20160418T121717Z:52204DE9EEFA6EB8:E7BF|SMID.1:Prism:94000507:20160506T095338Z:F184871DC4F23CB0:1F8C|20160506T122741Z|C9A5161F864E1978435A2CEAA611930F37824EDCE6252CEC|71CC

Table 3- VKLOADRESP.1 record

Field	Content
Record type	VKLOAD.RESP.1
KMCID.1, manufacturer	KMCID.1:Prism:K0001
GNT - generation time of the key pair	20160418T121717Z
Fingerprint	52204DE9EEFA6EB8
CRC of preceding record	E7BF
SMID.1, manufacturer, SMID	SMID.1:Prism:94000507
GNT - generation time of the key pair	20160506T095338Z
Fingerprint	F184871DC4F23CB0
CRC of preceding record	1F8C
TVP - time variant parameter from VKLOADREQ	20160506T122741Z
MACTAG - KMC key confirmation	C9A5161F864E1978435A2CEAA611930F37824EDCE6252CEC
CRC of entire record	71CC

16.1.2 Record type Key.1

KEY.1|A8556C52BA3345996C1551DC|ACT20160425T114321Z;BDT19930101T000000Z;CLM44fa0000;CLU0;DKG02;EXP20170506T215959Z;IUT20160903T102658Z;KCV17D819;KEN255;KRN1;KTC2;SBMFFFF;SGC0000112233;SGNSURELOAD KMC TEST VUDK 1;ULM1000;|849B9665C2BFFB75BF58629A0A057D528888559693DB293A|8F06

Table 4- Record type KEY.1

Field	Content
Record type	KEY.1
Unique NONCE	A8556C52BA3345996C1551DC
Activation date of the vending key	ACT20160425T114321Z
Base Date	BDT19930101T000000Z
Currency Credit Limit	CLM44fa0000

Cluster number	CLU0
DKGA number	DKG02
Key expiry date	EXP20170506T215959Z
Key issued until	IUT20160903T102658Z
Key Check Value	KCV17D819
Key Expiry Number	KEN255
Key Revision Number	KRN1
Key type	KTC2
Subclass bitmap	SBMFFFF
SGC Number	SGC0000112233
SGC Name	SGNSURELOAD KMC TEST VUDK 1
Unit credit limit	ULM1000
Protected (encrypted) vending key	849B9665C2BFFB75BF58629A0A057D528888559693DB293A
CRC of entire record	8F06

17 Appendix B – Vending Key attributes (normative)

The following table defines the attribute card names – and encoding of corresponding values – that may appear in the Attributes field of a WRAPPED-KEY record (section 7.5).

An Attributes field SHALL contain all cards (names) for which the Presence is indicated as “Required”, and MAY contain any names for which the Presence is “Optional”. The field MAY contain names other than those defined in this Appendix (the presence of which MUST be optional).

Table 5 - Vending Key Attributes

Name	Content type	Presence	Description
ACT	TIMESTAMP	Required	Activation Time: the date and time at which this Vending Key becomes active for the SGC. In spite of the implications of [IEC 62055-41] (section 6.5.2.5) the POS SHALL select the CurrentKey as the Vending Key in a supply group having the most recent Activation Time (that is, the highest Activation Time that is in the past); this behaviour is consistent with Legacy KMC practice and [SANS 1524-6-10] (in which this field is known as “EffectiveDate”).
BDT	TIMESTAMP	Required	Base Date: the date associated with a TID value of zero, as specified in [STS COP 402-1].
DKG	2D	Required	Decoder Key Generation Algorithm from [IEC 62055-41] section 6.1.4.
IUT	TIMESTAMP	Optional	Issued Until: a date and time after which the SM will prevent the key from being used for token encryption.
KEN	3D	Required	Key Expiry Number from [IEC 62055-41] section 6.1.10. The KEN must be in the range 0-255 and is interpreted relative to the Base Date (BDT).
KRN	1D	Required	Key Revision Number from [IEC 62055-41] section 6.1.10.
KTC	1D	Required	Key Type (KT) code from [IEC 62055-41] Table 24 (section 6.5.2.2.1), indicating whether the key is a VUDK, VCDK or VDDK.
SGC	10D	Required	Supply Group Code from [IEC 62055-41] section 6.1.6. This specification requires 10-digit SGCs; left-pad shorter SGCs with zero characters (“0”) to make then 10 digits long.
SGN	1-99P	Optional	Supply Group Name, a human-readable name for the supply group.

18 Appendix C – Record-in-email format (normative)

This format is intended to represent a single record (section 5.8) within the body of an e-mail message. The record is easily identified and extracted by a human operator or by software.

Given a record *REC* with record type *rectype*, a record-in-email is rendered as follows:

--STS:*rectype* BEGINS-- *REC wrapped to 64 characters or less per line* --STS:*rectype* ENDS--

The starting guard is the octet string x'2D2D5354533A ‖ *rectype* ‖ x'20424547494E532D2D, and the ending guard is x'2D2D5354533A ‖ *rectype* ‖ x'20454E44532D2D.

19 Appendix D – File-of-records format (normative)

This format is intended to represent one or more records (section 5.8) in a text file. The file is easily parsed by software and includes an *insecure* checksum to detect accidental data corruption.

A text file is an ordered sequence of lines. Each line contains only Printable ASCII characters and is terminated by a single End-Of-Line (EOL) character LF (x'0A, often given as '\n' in source code) or by the End-Of-File (EOF) condition. The EOL may be omitted from the last line of the file.

A file-of-records is a text file in which each line is either a record, a comment, or empty (whitespace). The last line in the file is a comment containing a BASE16-encoded SHA-1 [W:SHA-1] checksum over the preceding lines (including EOL characters), and must not have an EOL character.

A file-of-records is fully specified by the following production, given in Extended Backus-Naur Form [W:EBNF]:

```
File-of-records = Content, "#", Checksum ;
Content         = Line, LF, { Line, LF } ;
Checksum        = BASE16( SHA-1( Content ) ) ;
Line            = Record | Comment | Empty ;
LF              = x'0A ;
Record          = Printable, { Whitespace }
Comment         = "#", Printable ;
Empty           = { Whitespace } ;
Whitespace      = x'20 | x'08 | x'0D ;
```

Note that record lines may have trailing whitespace, which should be removed before parsing the record.

20 Appendix E – Summary of cryptographic primitives and standards (informative)

The following table summarises all cryptographic primitives (algorithms) employed by this specification, and indicates the standards to which they conform:

Table 6 - Cryptographic Primitives

Algorithm	Classification	Mode of Operation	Key	Key Length (bits)	Security-strength (bits)	Standards[1]
MISTY1	64-bit Block Cipher	ECB	Decoder Key (DK)	128	128	**ISO 18033-3**
CAST-128	64-bit Block Cipher	ECB	Decoder Key (DK)	128	128	**ISO 18033-3** RFC 2144
ECB mode for any block cipher algorithm	n-bit Block Cipher mode of operation	ECB	Determined by Block Cipher. ECB provides confidentiality only, with weaker guarantees for multi-block encryption.			**ISO 10116** NIST SP800-38A
KDF108-Feedback-HMAC-SHA-384	Symmetric Key Derivation Function	Feedback mode (iterated PRF over HMAC-SHA-384)	Vending Key (VK)	160	160 (up to 192 with 192-bit key)	NIST SP800-108
HMAC-SHA-384-192	Pseudorandom Function (PRF)	*N/A*	HMAC with SHA-384 has a maximum key length of 1023 bits, and a security-strength of up to 192 or 384 bits (depending on application).			RFC 4868 ***ISO 9797-2*** *FIPS PUB 198-1*
HMAC	Dedicated Message Authentication Code (MAC)	*Operates over a digest (hash) function*	Maximum key length depends on digest function. Security-strength depends on key length.			**ISO 9797-2** FIPS PUB 198-1 RFC 2104
SHA-384	Digest function (hash)	*N/A*	Non-keyed function. Security-strength is 192 bits for digital signatures and MAC, 384 bits for KDF.			**ISO 10118-3** FIPS PUB 180-4
AES-192	128-bit Block Cipher	CCM	Key Exchange Key (KEK)	192	192 (integrity limited to 128)	**ISO 18033-3** FIPS PUB 197
CCM mode for any block cipher algorithm	n-bit Block Cipher mode of operation	CCM (Nonce-based Authenticated Encryption with Additional Data)	Determined by Block Cipher. CCM provides confidentiality and integrity under the assumption that the nonce is not reused with a given key.			**ISO 19772** NIST SP800-38C RFC 3610
ECC CDH	Asymmetric key agreement primitive	Domain parameters: NIST P-384; KDF-X963	(d_{SM}, Q_{SM}) and (d_{KMC}, Q_{KMC})	384	192	**ISO 11770-3** ANSI X9.63 NIST SP800-56A SEC 1
1-Pass Unified Model C(1e, 2s)	Asymmetric key agreement scheme	Operates over ECC CDH primitive	2 static (as for ECC CDH) plus 1 ephemeral for SM: (d_E, Q_E)	384	192	**NIST SP800-56A** ANSI X9.63 ISO 11770-3

[1] Normative standards are in **bold**.

Algorithm	Classification	Mode of Operation	Key	Key Length (bits)	Security-strength (bits)	Standards[1]
ECDSA	Digital signature	Domain parameters: NIST P-384; SHA-384	(d_{MAN}, Q_{MAN})	384	192	**ISO 14888-3** ANSI X9.62 FIPS PUB 186-3 SEC 1
P-384 for any ECC operation	ECC Domain Parameters	P-384	Also known as "ansix9p384r1" and "secp384r1". ECC operations in this domain can provide up to 192 bits of security.			**FIPS PUB 186-3** ANSI X9.62 SEC 2
KDF-X963-SHA-384	Key Derivation Function (KDF)	Counter mode (iterated PRF over SHA-384)	Shared Secret from ECC CDH	Depends on ECC CDH (minimum 192 bits entropy required)	192 bits	**ISO 11770-3** ANSI X9.63 SEC 1

The following table indicates alignment of cryptographic primitives employed by this specification with various standards bodies and projects, with respect to the context or purpose of use:

Table 7 - Alignment of cryptographic primitives

Algorithm	ISO	NIST	NISTIR 7628 Smart Grid[1]	SP800-152 Federal KMC	*Others*
MISTY1, ECB *for token encryption*	**ISO/TR 14742** ISO 18033-3 ISO 10116	*MISTY1 and CAST-128 are not approved by any NIST or FIPS standard.*			RFC 2994 Approved by NESSIE & CRYPTREC
CAST-128, ECB *for token encryption*	**ISO/TR 14742** ISO 18033-3 ISO 10116				RFC 2144 Approved by CSEC
KDF108-Feedback-HMAC-SHA-384 with LVCONCAT *for symmetric key derivation*	*No relevant standard* HMAC: ISO 9797-2 and **ISO/TR 14742** SHA-384: ISO 10118-3 and **ISO/TR 14742**	**NIST SP800-131A** NIST SP800-108 FIPS PUB 198-1 FIPS PUB 180-4 Formatting function complies fully	Approved beyond 2030	Exceeds "Augmented" security; not interoperable	Equivalent to RFC 5869; not interoperable
AES-192, CCM *for authenticated encryption and key wrapping*	ISO 18033-3 ISO 19772 AES: **ISO/TR 14742**	**NIST SP800-131A** FIPS PUB 197 NIST SP800-38C	Approved beyond 2030	Exceeds "Augmented" security; not interoperable	ANSI X9.102 Approved by NESSIE & CRYPTREC
ECC CDH C(1e, 2s) *for asymmetric shared secret agreement*	ISO 11770-3	**NIST SP800-131A** NIST SP800-56A (Set ED)	Approved beyond 2030, 192-bit security	Exceeds "Augmented" security; not interoperable	NSA Suite B ANSI X9.63 SEC 1
ECDSA *For digital signature in PK certificates*	**ISO/TR 14742** ISO 14888-3	**NIST SP800-131A** FIPS PUB 186-3	Approved beyond 2030	Exceeds "Augmented"; meets 'Desirable'	NSA Suite B ANSI X9.62 SEC 1

[1] The NIST 7628 requirements for approval beyond 2030 are based on NIST SP800-57 and NIST SP800-131.

Algorithm	ISO	NIST	NISTIR 7628 Smart Grid[1]	SP800-152 Federal KMC	*Others*
NIST P-384 domain parameters	**ISO/TR 14742** *No ISO standard specifies curves*	**NIST SP800-131A** FIPS PUB 186-3	Approved beyond 2030	Exceeds "Augmented" security; not interoperable	NSA Suite B ANSI X9.62 SEC 2
KDF-X963-SHA-384 with LVCONCAT *for key derivation from asymmetrically shared secret*	ISO 11770-3	**NIST SP800-131A** NIST SP800-135 or NIST SP800-131A; Meets SP800-56A Set ED targets.	Approved beyond 2030	*Non-compliant: only NIST concatenation KDF permitted*	ANSI X9.63 SEC 1
Unified Model key confirmation (with HMAC-SHA-384-192 and LVCONCAT)	Conforms to ISO 11770-3	Partially conforms but not interoperable; Meets SP800-56A Set ED targets	*No relevant guidance*	Meets "Augmented" requirements	Partially conforms to ANSI X9.63; standard is unclear on key confirmation for C(1e, 2s)

Comments:

- [ISO/TR 14742] and [NIST SP800-131A] are 'super-standards' that recommend – with reference to other standards – cryptographic algorithms and key lengths that are appropriate for the foreseeable future.
    - [ISO/TR 14742] provides recommendations for the financial services industry. The body of the standard does not cover key establishment algorithms, but Annex A cites key establishment mechanisms in [ISO 11770-3]. The standard does not cover authenticated encryption modes of operation.
    - [NIST SP800-131A] specifies NIST Approved algorithms that may be implemented in a [FIPS PUB 140-2] certified HSM, and indicates the permitted periods of use for these algorithms and associated key lengths. The standard does not recommend ECC curves or cover key confirmation in key agreement algorithms, but does approve the schemes in [NIST SP800-56A], and that standard in turn recommends the curves in [FIPS PUB 186-3].
- An algorithm is fully aligned with the cited standard(s) unless otherwise indicated. Full alignment includes security equivalence and interoperability.
- An algorithm may *conform* to a standard without being fully interoperable. This usually occurs when this specification has a higher security target than the standard (as with NIST SP800-152) or the standard specifies that formatting of input fields is application specific (as with KDFs and key confirmation).

21 Appendix F – Summary of functions (informative)

A reference list of functions defined elsewhere in this document:

- BCD(Decimal String) → Octet String | Error
- BASE16(Octet String) → Hexadecimal String
- BASE16-DECODE(Hexadecimal String) → Octet String | Error
- Integer-to-Octet-String(Integer, MaxInteger) → Octet String | Error
- Octet-String-to-Integer(Octet String, MaxInteger) → Integer | Error
- Field-Element-to-Octet-String$_{Domain}$(Field Element) → Octet String | Error
- Octet-String-to-Field-Element$_{Domain}$(Octet String) → Field Element | Error
- Point-to-Octet-String$_{Domain}$(Point) → Octet String | Error
- Octet-String-to-Point$_{Domain}$(Octet String) → (x_P,y_P) not necessarily a valid Point | Error
- CRC16-MODBUS(Octet String) → 16-bit Big Endian integer
- LVCONCAT(I_1, I_2, …, I_n), I_i an Octet String, OctetLen(I_i) ≤ 255, n ≤ 255 → Octet String | Error
- DFCONCAT(DELIM, I_1, I_2, …, I_n), DELIM 1P, I_i Printable → Printable ASCII String | Error
- DFPARSE(DELIM, Octet String) → O_1, O_2, …, O_n, O_i Printable | Error
- BUILD-RECORD(rectype, delim, n, I_1, I_2, …, I_n), rectype IDENT, I_i Printable → Printable ASCII String | Error
- PARSE-RECORD(rectype, delim, n, Octet String), rectype IDENT, → O_1, O_2, …, O_n, O_i Printable | Error
- AES-192-CCM$_{ENC}$(Key, Nonce, Additional, Plaintext) → Ciphertext | Error; *all Octet String*
- AES-192-CCM$_{DEC}$(Key, Nonce, Additional, Ciphertext) → Plaintext | Error; *all Octet String*
- SHA-384(Octet String) → Digest (*Octet String*)
- HMAC-SHA-384-192(Key, Text) → MAC; *all Octet String*
- KDF-X963-SHA-384(SharedSecret, SharedInfo, keydatalen) → Key Material (*Octet String*)
- ECC-CDH$_{P\text{-}384}$(d_A in [1,n-1], Q_B a Point) → SharedSecret (*Octet String*)
- ECDSA-SIGN$_{P\text{-}384,SHA\text{-}384}$($d_A$ in [1, n-1], M an Octet String) → (r, s) both in [1, n-1] | Error
- ECDSA-VERIFY$_{P\text{-}384,SHA\text{-}384}$($Q_B$ a Point, M an Octet String, (r,s) a Signature) → “valid” | “invalid” | Error
- GENERATE-KEY() → ECC Key Pair (d_A in [1,n-1], Q_A a Point)

- VALIDATE-KEY(Q_B a Point) → TRUE | Error
- CAST-128_{ENC}(Key, Plaintext) → Ciphertext; *all Octet String*
- CAST-128_{DEC}(Key, Ciphertext) → Plaintext; *all Octet String*
- MISTY1_{ENC}(Key, Plaintext) → Ciphertext; *all Octet String*
- MISTY1_{DEC}(Key, Ciphertext) → Plaintext; *all Octet String*
- HMAC-DKGA(VK, SGC, KT, KRN, MeterPAN, EA, TI) → Key
- KDF108-Feedback-HMAC-SHA-384(DerivationKey, OtherInfo, keydatalen) → Key Material

22 Appendix G – Summary of required Codes of Practice and Registries (informative)

The STSA SHOULD provide Codes of Practice for:

- The security requirements for an SM (see section 9.1).
- The security requirements for a KMC HSM (see section 10.1).
- The requirements for approving SM hardware and firmware (see section 9.1).

The STSA SHOULD provide registry services for:

- Manufacturer names (see section 8).
- KMC names (see section 10.2).
- Approved HWIDs (see section 10.2).
- Approved FWIDs (see section 10.2).

KMC standards to be developed by the STSA SHALL include:

- A procedure for SM Manufacturers to publish their public key certificates to KMCs in a trusted manner (section 8).
- A procedure for KMCs to publish their public keys to SM Operators in a trusted manner (section 10.3).
- Procedures or procedural requirements to handle the compromise of KMC keys (d_{KMC}, SMK) or of Vending Keys (section 14.3.2).

23 Appendix H – Implementation guidance (informative)

This section provides miscellaneous guidance for implementing this standard.

- The ASCII character '+' (x'2B) is suggested as a field delimiter when implementations must concatenate fields (comprising data types or structures defined in this standard) as a consequence of implementation. The '+' character is safe for many printable encodings and in URLs; other characters may be less safe or may conflict with the use of delimiters in this standard.

24 Appendix I - Key Agreement Scheme - worked example (informative)

The key agreement scheme between HSM and KMS is complex. A full worked example using static test vectors is available for manufacturers of HSM devices as a reference for implementation of the Key Agreement Scheme specified in this document.

This worked example may be found in the STS600-9-1 document. Tests using ephemeral data may only be done using the STSA test KMS.