INTERNATIONAL STANDARD

ISO 11568

First edition
2023-02

Financial services — Key management (retail)

Services financiers — Gestion de clés (services aux particuliers)

Reference number
ISO 11568:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 68, *Financial services*, Subcommittee SC 2, *Financial services, security*.

This document cancels and replaces the former ISO 11568 series, which has been technically revised.

The main changes are as follows:

— all parts of the series combined into a single document;

— fixed key no longer included in the permissible methods of transaction key management;

— required key replacement policy (see 4.13) added;

— cleartext key injection removed;

— AES DUKPT introduced as a key management method.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Retail financial transactions are often transmitted over potentially non-secure channels, which, if exploited, can result in fraud. The vast range in value and volume of such transactions exposes participants to severe risks, which can be uninsurable. To protect against these risks, many institutions are employing encryption. The encryption algorithms used are in the public domain. The security and reliability of any process based on these algorithms is directly dependent on the protection afforded to secrets called cryptographic keys.

This document describes requirements and provides guidance for the secure management of cryptographic keys used to protect sensitive information in a retail financial services environment, for example in messages between a card acceptor and an Acquirer. Typical services in the retail financial services domain include point-of-sale (POS) debit and credit authorizations and automated teller machine (ATM) transactions. While it is designed with these environments in mind, it may also be used in unrelated applications. For example, such keys could be used for:

— encrypting Personal Identification Numbers (PIN) (see ISO 9564-1);

— authenticating messages;

— encrypting other data;

— encrypting or deriving cryptographic keys;

— automated symmetric key distribution using asymmetric techniques.

Financial services — Key management (retail)

1 Scope

1.1 General

This document describes the management of symmetric and asymmetric cryptographic keys that can be used to protect sensitive information in financial services related to retail payments. The document covers all aspects of retail financial services, including connections between a card-accepting device and an Acquirer, between an Acquirer and a card Issuer, and between an ICC and a card-accepting device. It covers all phases of the key life cycle, including the generation, distribution, utilization, archiving, replacement and destruction of the keying material. This document covers manual and automated management of keying material, and any combination thereof, used for retail financial services. It includes guidance and requirements related to key separation, substitution prevention, identification, synchronization, integrity, confidentiality and compromise, as well as logging and auditing of key management events.

Requirements associated with hardware used to manage keys have also been included in this document.

1.2 Scope exclusions

This document does not specifically address internet banking services offered by an Issuer to their own customers through that financial institution's website or applications.

This document does not address using asymmetric keys to encrypt the Personal Identification Number (PIN) or any other data and does not address asymmetric keys managed with asymmetric keys.

This document is not intended to apply to the management of the keys installed in an ICC during manufacturing or the initial key established in an ICC during card personalization.

This document is not intended to address post-quantum encryption considerations. Key management using quantum technologies is out of scope of this document.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9797 (all parts), *Information technology — Security techniques — Message Authentication Codes (MACs)*

ISO/IEC 11770 (all parts), *Information security — Key management*

ISO 13491 (all parts), *Financial services — Secure cryptographic devices (retail)*

ISO 16609, *Financial services — Requirements for message authentication using symmetric techniques*

ISO/IEC 18031, *Information technology — Security techniques — Random bit generation*

ISO/IEC 18032, *Information security — Prime number generation*

ISO/IEC 18033 (all parts), *Information security — Encryption algorithms*

ISO/IEC 19592-2, *Information technology — Security techniques — Secret sharing — Part 2: Fundamental mechanisms*

ISO/IEC 19772, *Information security — Authenticated encryption*

ISO 20038, *Banking and related financial services — Key wrap using AES*

ISO 21188:2018, *Public key infrastructure for financial services — Practices and policy framework*

ANSI X9.63, *Public Key Cryptography for the Financial Services Industry — Key Agreement and Key Management Using Elliptic Curve-Based Cryptography*

ANSI X9.143, *Retail Financial Services — Interoperable Secure Key Block Specification*

RFC 3647, *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, Internet Request for Comments 3647, S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu, November 2003*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
Acquirer
institution (or its agent) which acquires from the card acceptor the data relating to the *transaction* (3.84) and initiates the data into an *Interchange* (3.47) system

[SOURCE: ISO/IEC 7812-1:2017, 3.1]

3.2
Advanced Encryption Standard
AES
16-byte block *cipher* (3.3)

Note 1 to entry: This is defined in ISO/IEC 18033-3.

3.3
algorithm
specified mathematical process for computation or set of rules which, if followed, will give a prescribed result

[SOURCE: ISO 16609:2022, 3.1]

3.4
archived key
inactive *cryptographic key* (3.28) that is being stored in a secure manner for a non-operational purpose

3.5
asymmetric algorithm
cryptographic algorithm (3.27) that uses two related keys, a *public key* (3.71) and a *private key* (3.69), where the two keys have the property that, given the public key, it is *computationally infeasible* (3.25) to derive the private key

3.6
asymmetric cryptosystem
cryptosystem using *asymmetric algorithms* (3.5)

3.7
authentication
provision of assurance that a claimed characteristic of an entity is correct

[SOURCE: ISO/IEC 27000:2018, 3.5]

3.8
authentication element
message element that is to be protected by *authentication* (3.7)

[SOURCE: ISO 16609:2022, 3.12, modified — Term revised.]

3.9
Base Derivation Key
BDK
key used in *derivation* (3.32) to generate *initial DUKPT keys* (3.45) for installation into transaction originating *secure cryptographic devices* (3.75) and for transaction processing

3.10
BDK ID
32-bit value that identifies the *Base Derivation Key* (3.9)

Note 1 to entry: This was formerly known as the *Key Set Identifier* (*KSI*) (3.57) in the TDEA DUKPT specification.

3.11
card acceptor
party accepting the card for the purpose of presenting transaction data to an *Acquirer* (3.1) or intermediary facilitating the transaction flow

[SOURCE: ISO/IEC 7812-1:2017, 3.3]

3.12
certificate
digitally signed statement that binds the value of a public key to the identity of the person, device or service that holds the corresponding private key

[SOURCE: ISO 20415:2019, 3.15, modified — Term revised.]

3.13
certificate authority
CA
entity that vouches for the binding between a device's identity, its public key and associated keying material

[SOURCE: ISO/IEC/IEEE 8802-11:2022, definition modified.]

3.14
certificate authority system
CA system
infrastructure required to manage, maintain and secure the key pairs and certificates of the *certificate authority* (3.13)

Note 1 to entry: A CA system will typically include one or more *Hardware Security Modules* (3.42), firmware, computer equipment, operating systems and software

3.15
certificate policy
CP
named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements

[SOURCE: ISO 21188:2018, 3.13]

3.16
certificate practice statement
CPS
statement of the practices which a *certificate authority* (3.13) employs in issuing *certificates* (3.12) and which defines the equipment, policies and procedures the certificate authority uses to satisfy the requirements specified in the *certificate policies* (3.15) that are supported by it

3.17
certificate subject
entity identified in a *certificate* (3.12)

EXAMPLE *Secure cryptographic device* (3.75).

3.18
chain of custody
demonstrable possession, movement, handling and location of material from one point in time until another

[SOURCE: ISO/IEC 27050-1:2019, 3.1]

3.19
check value
key check value
KCV
component check value
CCV
non-secret value that is cryptographically related to the key (or component) and is used to verify that the underlying value is as expected

Note 1 to entry: It is possible for different keys or components to have the same check value.

3.20
cipher
method for the transformation of data in order to hide its information content, prevent its undetected modification and/or prevent its unauthorized use

3.21
ciphertext
data which has been transformed to hide its information content

[SOURCE: ISO/IEC 18033-1:2021, 3.7]

3.22
cleartext
plaintext
unencrypted information

[SOURCE: ISO/IEC 18033-1:2021, 3.20]

3.23
communicating pair
two parties (usually institutions) sending and receiving transactions

Note 1 to entry: This includes alternate processing sites either owned or contracted by either communicating party.

3.24
compromise
<cryptography> breach or failure of the security of a process or system used to protect the confidentiality or integrity of sensitive information

Note 1 to entry: Compromise includes situations in which either unauthorized disclosure of sensitive information could have occurred or appropriate control is not demonstrable.

3.25
computationally infeasible
property that a computation is theoretically achievable but is not feasible in terms of the time or resources required to perform it

3.26
credentials
identification data for an entity, incorporating at a minimum the entity's distinguished name and public key

3.27
cryptographic algorithm
algorithm (3.3) for the transforming of data using a *cryptographic key* (3.28)

Note 1 to entry: Data transformation includes operations such as *encryption* (3.40), *decryption* (3.31), synchronized generation of keying material and computation or verification of a digital signature or Message Authentication Code.

3.28
cryptographic key
key
sequence of bits that determine the outcome of a cryptographic operation

Note 1 to entry: See cryptographic algorithm for further details.

Note 2 to entry: Examples of cryptographic operations include encryption, decryption, check function computation, derivation, signature generation or verification, and authentication.

3.29
cryptographic strength
strength
computational cost of the most effective known attack against a given cryptographic algorithm and key size

Note 1 to entry: The cryptographic strength is usually a function of key size, but different algorithms can have different cryptographic strengths even though their key sizes are the same.

Note 2 to entry: The strength of a given algorithm and key size can vary depending on the use of the algorithm. For example, uses with few or no plaintext-ciphertext pairs could be more resistant to attack than uses with many pairs under a given key.

3.30
cryptoperiod
defined period of time during which a specific *cryptographic key* (3.28) is authorized for use or during which the cryptographic keys in a given system may remain in effect

[SOURCE: ISO 16609:2022, 3.9]

3.31
decryption
process of transforming *ciphertext* (3.21) data into *cleartext* (3.22) data

3.32
derivation
cryptographic transformation process that is used to derive one value from another value

Note 1 to entry: This is typically used to produce new *cryptographic keys* (3.28), which can potentially be used for different purposes, from a single key.

3.33
Derivation Identifier
Derivation ID
DID
32-bit value that identifies the specific *initial DUKPT key* (3.45) derived using the *Base Derivation Key* (3.9)

Note 1 to entry: This was formerly known as the 19-bit value Device ID (DID) or TRSM ID in the TDEA DUKPT specification.

3.34
derivation key
cryptographic key (3.28) that is used to cryptographically compute another key using *derivation* (3.32)

3.35
derived unique key per transaction
DUKPT
key management method that uses a unique key for each *transaction* (3.84) and prevents the disclosure of any past key used by the *transaction originating SCD* (3.86)

Note 1 to entry: The unique *transaction keys* (3.85) are derived from a *Base Derivation Key* (3.9) and non-secret data transmitted as part of each transaction.

3.36
digital certificate
asymmetric cryptosystem that provides for the creation and subsequent verification of *digital signatures* (3.37)

3.37
digital signature
cryptographic transformation of data which, when associated with a data unit and accompanied by the corresponding public-key certificate, provides the services of origin authentication, data integrity and signer non-repudiation

3.38
double-length TDEA key
TDEA key having a length of 128 bits, consisting of 112 key bits and 16 parity bits, which is typically represented in 32 hexadecimal digits

3.39
dual control
process of utilizing two or more separate individuals operating in concert to protect sensitive functions or information whereby no single individual is able to use the function or access all the information alone

Note 1 to entry: A *cryptographic key* (3.28) is an example of the type of material protected using dual control.

Note 2 to entry: For protecting cryptographic keys and other sensitive data, this concept is closely related to *split knowledge* (3.77).

[SOURCE: ISO 9564-1:2017, 3.10, modified — Definition revised and notes to entry added.]

3.40
encryption
encipherment
process of transforming *cleartext* (3.22) data into *ciphertext* (3.21) data for the purpose of security or privacy

3.41
exclusive-or
XOR
mathematical operation defined as: 0 XOR 0 = 0 0 XOR 1 = 1 1 XOR 0 = 1 1 XOR 1 = 0

Note 1 to entry: Equivalent to binary addition without carry (modulo-2 addition).

3.42
Hardware Security Module
HSM
secure cryptographic device (3.75) that provides a set of secure cryptographic services including, but not limited to, key generation, cryptogram creation, PIN translation and certificate signing

[SOURCE: ISO 13491-1:2016, 3.23]

3.43
hash function
one-way function that maps a set of strings of arbitrary length on to a set of fixed-length strings of bits

Note 1 to entry: The output is generally relatively small.

Note 2 to entry: A collision-resistant hash function has the property that it is computationally infeasible to construct distinct inputs that map to the same output.

3.44
independent communication
process that allows an entity to counter-verify the correctness of a credential and identification documents prior to producing a certificate

EXAMPLE Call-back, visual identification.

3.45
initial DUKPT key
IK
unique *cryptographic key* (3.28) loaded into a *secure cryptographic device* (3.75) that has been derived from a *Base Derivation Key* (3.9)

3.46
Initial Key ID
64-bit value that identifies the specific *initial DUKPT key* (3.45) derived under the *Base Derivation Key* (3.9)

Note 1 to entry: It is a concatenation of the *BDK ID* (3.10) and the *Derivation ID* (3.33)

3.47
Interchange
mutual acceptance and exchange of messages between institutions for card payment transactions

3.48
Issuer
institution holding the account identified by the Primary Account Number (PAN)

Note 1 to entry: See Annex J for additional information on the role an Issuer plays in the payment environment.

3.49
key agreement
process of establishing a shared secret key between entities in such a way that neither of them can predetermine the value of that key

Note 1 to entry: By predetermine, it is meant that neither entity A nor entity B can, in a computationally efficient way, choose a smaller key space and force the computed key in the protocol to fall into that key space.

[SOURCE: ISO/IEC 11770-4:2017, 3.13]

3.50
key component
component
one of at least two values that are combined using XOR to form a symmetric *cryptographic key* (3.28)

Note 1 to entry: Each component has the same format (including length) as the cryptographic key.

3.51
key distribution host
KDH
host distributing keys in the *SKDAT* (3.79) key distribution methodology

Note 1 to entry: See Annex K for more details on the role a KDH plays in SKDAT.

3.52
key encryption key
KEK
key used exclusively to encrypt and decrypt other keys

3.53
key receiving device
KRD
device receiving keys in the *SKDAT* (3.79) key distribution methodology

3.54
key separation
method for ensuring that a key is used for only its intended purpose

3.55
Key Serial Number
KSN
concatenation of the *Initial Key ID* (3.46) and the transaction counter used in *DUKPT* (3.35)

3.56
key set
group of keys that are all determined by a common cryptographic procedure and differentiated by non-secret input such that knowledge of one key does not disclose any other key in the group

3.57
Key Set Identifier
KSI
non-secret value that uniquely identifies a *key set* (3.56)

3.58
key share
result of dividing a *cryptographic key* (3.28) into some number (n) of pieces (shares), such that a designated minimum number (m) of pieces are required to reconstitute the key

Note 1 to entry: Each key share is constructed in such a manner that access to fewer than m shares does not disclose any information about the key. In all cases, m is greater than 1 and less than or equal to n.

Note 2 to entry: A secret sharing scheme involving key shares is often referred to as an "m of n scheme".

3.59
keying material
data that comprise a complete *cryptographic key* (3.28) and its relevant metadata

EXAMPLE Keys, attributes, initialization vectors.

3.60
master key
highest level of key encrypting key in a hierarchy of *key encryption keys* (3.52) and *transaction keys* (3.85)

3.61
message authentication
verification that a message was sent by the purported originator to the intended recipient and that the message was not changed in transit

[SOURCE: ISO/IEC 20944-1:2013, 3.11.1.9]

3.62
Message Authentication Code
MAC
cryptographic value used to confirm that the message came from the stated sender and has not been changed in transit

3.63
node
point in a network that does some form of processing of data

EXAMPLE Terminal, Acquirer, switch.

3.64
non-repudiation
service that provides verifiable evidence to substantiate integrity and origin of data, thereby eliminating successful deniability

3.65
parity
result of a calculation of the number of '1' bits in a string of '0' and '1' bits that indicates whether the number of '1' bits is odd or even

3.66
payment instrument
physical payment card, electronic equivalent or other electronic instrument or order used for the transmission or payment of money, sold or issued to one or more persons, whether or not the instrument is negotiable

Note 1 to entry: This does not include any credit card voucher, any letter of credit or any instrument that is redeemable by the Issuer in goods or services.

3.67
Personal Identification Number
PIN
string of numeric digits established as a shared secret between the account owner and the *Issuer* (3.48), for subsequent use to validate authorized card usage

[SOURCE: ISO 9564-1:2017, 3.19, modified]

3.68
Primary Account Number
PAN
assigned number, composed of an Issuer identification number, an individual account identification and an accompanying check digit that identifies the card Issuer and account owner (who is the cardholder for physical cards)

Note 1 to entry: The accompanying check digit is specified in ISO/IEC 7812-1.

[SOURCE: ISO 9564-1:2017, 3.22, modified]

3.69
private key
key in an asymmetric key pair which is known only by that entity

3.70
pseudo-random process
process that produces a value which is statistically random and essentially unpredictable even though it was generated by a deterministic algorithm

3.71
public key
key of an entity's asymmetric key pair which can usually be made public without compromising security

[SOURCE: ISO/IEC 11770-1:2010, 2.36]

3.72
random
value in a set that has an equal probability of being selected from the total population of possibilities, hence is unpredictable

3.73
recipient
person, institution or other party that is responsible for and authorized to receive a message or package

3.74
replay
process of sending a message which contains all or part of a previously sent message, as a method of perpetrating a fraud

3.75
secure cryptographic device
SCD
device that provides physically and logically protected cryptographic services and storage and which can be integrated into a larger system such as an ATM or POS terminal

EXAMPLE entry device (PED), Hardware Security Module (HSM).

[SOURCE: ISO 13491-1:2016, 3.28]

3.76
sender
person, institution or other entity transmitting a message or package

3.77
split knowledge
condition under which two or more individuals separately and confidentially have information that, individually, convey no knowledge of the resulting combined information

EXAMPLE *Components* (3.50) of a *cryptographic key* (3.28) managed under strict custody separation such that no one person can gain access to the entire set.

[SOURCE: ISO 9564-1:2017, 3.28, modified — Definition revised and example added.]

3.78
symmetric key
secret *cryptographic key* (3.28) that is used in a symmetric *algorithm* (3.3)

3.79
symmetric key distribution using asymmetric techniques
SKDAT
key distribution method that uses asymmetric cryptography to protect the keys during transport

Note 1 to entry: See Annex K.

3.80
switch
node that can route data from a node to other nodes

3.81
tamper evident and authenticable bag
TEA bag
one-time use packaging that is designed in such a way as to make it infeasible to access the contents without detection and has a pre-printed unique identifier that cannot be changed without detection to allow for authentication

Note 1 to entry: The form of the packaging is not necessarily restricted to a bag.

3.82
tampering
unauthorized modification that compromises the security properties of a device or security container

EXAMPLE 1 Insertion of active or passive tapping mechanisms into a device to determine, record or modify secret data.

EXAMPLE 2 An attempt to penetrate or open a security container, TEA bag or device.

3.83
terminal
device or system that initiates a *transaction* (3.84) and is, contains or interfaces with a *transaction-originating SCD* (3.86) for cryptographic functions

3.84
transaction
series of messages to perform a predefined function

EXAMPLE Payment transaction.

3.85
transaction key
key used to cryptographically protect the transaction data elements between nodes

Note 1 to entry: If more than one key is used for different cryptographic functions, each key can be a variant or derivative of the transaction key.

Note 2 to entry: A transaction key is sometimes referred to as a data key, communications key, session key or working key.

3.86
transaction-originating SCD
secure cryptographic device (3.75) that is, is integrated in or is used by the *terminal* (3.83) that initiates the financial *transaction* (3.84)

EXAMPLE PIN entry device.

Note 1 to entry: An intermediate device that processes the transaction, such as a *Hardware Security Module* (3.42), is not a transaction-originating SCD.

3.87
Triple Data Encryption Algorithm
TDEA
symmetric *algorithm* (3.3)

Note 1 to entry: The TDEA is specified in the ISO/IEC 18033 series.

3.88
triple-length TDEA key
Triple Data Encryption Algorithm (3.87) key having a length of 192 bits, consisting of 168 key bits and 24 parity bits, that is typically represented in 48 hexadecimal digits

3.89
verification
process of associating and/or checking a unique characteristic

4 Key management requirements

4.1 General

4.1.1 Key management strategy

Key generation, storage, loading, transport, use and destruction are points at which a key management policy is applied to ensure the secrecy of the keys. For an illustration of the symmetric key life cycle, see Annex D; for the asymmetric key life cycle, see Annex E.

The impact of a compromise of a given key should be taken into consideration when deciding the policy for management.

It is necessary to employ mechanisms that detect or prevent the unauthorized use of a key, the use of modified or substituted keys or the replay of keys.

This subclause includes general requirements that apply to the management of keys.

4.1.2 Dual control and split knowledge of secret or private keys

The management of cleartext components or shares of symmetric (secret) and asymmetric private keys shall ensure dual control and split knowledge of the resulting key. Throughout the life cycle of a key, an individual shall not have (or have had) access to all cleartext components or a sufficient number of shares to form the key. Each person or group of persons (e.g. custodian) responsible for each key component or share shall have received adequate training and shall have signed an acknowledgement of their responsibilities to keep secret the key component or share entrusted to them.

The chain of custody for each cleartext component or share shall be documented and such documentation shall be maintained.

Access to fewer than the number of shares required to reconstruct the plain text symmetric or private key or components required to reconstruct a symmetric key shall give no information about the overall key.

A cleartext key share or component of a symmetric or private key shall be accessible only to the person or group of persons to whom it has been entrusted and only for the minimum duration required.

4.1.3 Permissible key forms

4.1.3.1 Permissible symmetric key forms

Symmetric cryptographic keys shall only exist in one or more of the following forms:

a) in a secure cryptographic device (SCD) as specified in 4.2;

b) if encrypted and outside an SCD:

   1) as a cryptogram of the key that has been created inside an SCD using an AES or TDEA key encrypting key (see 4.1.5 for applicable key strength and 4.5 for key block requirements);

   2) as cryptograms of key shares or key components, where the cryptograms have been created inside an SCD using an AES or TDEA key encrypting key (see 4.1.5 for applicable key strength and 4.5 for key block requirements);

   3) as cryptograms of the key, key shares or key components that have been created inside an SCD using an asymmetric algorithm listed in Annex F.

c) if not encrypted and outside of an SCD:

   1) as two or more full-length key components that are managed in accordance with 4.1.2, thereby achieving dual control and split knowledge of the key;

   2) as two or more key shares that are created using a secret sharing method which is perfectly information-theoretically confidential, is defined in ISO/IEC 19592-2 and managed in accordance with 4.1.2, thereby achieving dual control and split knowledge of the key.

A component shall only be associated with a single key, except by chance.

A key encrypting key that protects a large number of keys, such as an Acquirer or Issuer top-level key that is managed as components or shares, should be managed as at least three separately managed components or shares, so that it requires at least three people to load the key.

4.1.3.2 Permissible asymmetric key forms

4.1.3.2.1 Asymmetric private key forms

Asymmetric private keys shall only exist in one or more of the following forms:

a) within an SCD as specified in 4.2.

b) if encrypted and outside an SCD (including transport between SCDs):

   1) as a cryptogram of the key that has been created inside an SCD using an AES or TDEA key encrypting key (see 4.1.5 for applicable cryptographic strength and 4.5 for key block requirements);

   2) as cryptograms of key shares, where the cryptograms have been created inside an SCD using an AES or TDEA key encrypting key (see 4.1.5 for applicable key strength and 4.5 for key block requirements);

   3) as cryptograms of the key or key shares that have been created inside an SCD using RSA OAEP or ECIES as defined in ISO/IEC 19592-2;

c) if not encrypted and outside of an SCD, only as two or more key shares that are created using a secret sharing method which is perfectly information-theoretically confidential, is included in ISO/IEC 19592-2 and managed in accordance with 4.1.2, thereby achieving dual control and split knowledge of the key.

4.1.3.2.2 Asymmetric public key forms

In an asymmetric cryptosystem there is no secrecy requirement for the storage of the public key, but authenticity and integrity of this key shall be ensured before use, preventing substitution or alteration of the public key or associated information without detection.

One or more of the following techniques shall be used to ensure public key integrity:

— sign the public key and associated data using a digital signature system, thereby creating a public key certificate in accordance with 4.3 and 4.4;

— create a MAC for the public key and associated data, using an algorithm defined by ISO 16609 and a key used only for this purpose;

— depend on protections of an SCD (see the ISO 13491 series);

— use the key verification techniques as defined in 4.8.3 (when the public key is stored in plain text).

4.1.4 Logging

4.1.4.1 General

All instances in which cleartext key components or shares are created, received, accessed, transported, loaded, stored or destroyed shall be recorded. A policy that defines the length of time such records are maintained shall exist and be in use. Consideration should be given to the possible need to inspect those records for a finite length of time after the key is removed from service. At a minimum, records kept by each party managing the key shall reflect the date, time, person(s) involved, activity type (e.g. generation, destruction, shipment), reason for activity, SCD Identifier (if applicable) and TEA bag number(s) (if applicable).

Except in the case of intermediate or ephemeral keys that exist only within a specific SCD and only for the length of time necessary to execute a cryptographic function, instances where cryptographic keys are created, transported, received, replaced or destroyed without manual intervention (e.g. dynamic key exchange) should be logged in such a way as to provide evidence that the event occurred. Similarly, instances where cryptograms are created, received, transported, loaded or stored with manual intervention should be logged in such a way as to provide evidence that the event occurred. Such logging may be done outside the SCD.

4.1.4.2 Asymmetric keys

All operations involving private keys shall be recorded.

The integrity of the logs shall be preserved.

For CA operations involving private keys, logs shall include reference to all hardware used in support of these activities. Hardware reference shall include SCD serial numbers and a means to distinguish make and model as a minimum. Other device attributes may also be recorded. The logs may be manual and/or electronic and the information may be contained in a combination of logs.

Operations involving the public key shall be recorded as indicated in Table 1.

Table 1 — Public key recording matrix

	Created	Signed	Installed as trusted key	KDH bind	KDH unbind	Destroyed
CA	✓	✓	✓			✓
KDH	✓	✓	✓	✓	✓	✓
NOTE Events related to installation or removal of a KDH public key in KDH bind and KDH unbind are typically logged at the KDH rather than at the KRD, even though the KRD is the device that is receiving or removing the KDH public key.						

Table 1 *(continued)*

	Created	Signed	Installed as trusted key	KDH bind	KDH unbind	Destroyed
KRD	✓	✓	✓			
NOTE Events related to installation or removal of a KDH public key in KDH bind and KDH unbind are typically logged at the KDH rather than at the KRD, even though the KRD is the device that is receiving or removing the KDH public key.						

4.1.4.3 SCD hardware logging

Chain of custody logging shall be maintained for all SCDs, excluding access related to the use of deployed transaction-originating SCDs.

Approved settings for user-defined parameters and configurations for HSMs shall be defined and logs reflecting those chosen during configuration activities shall be maintained.

4.1.5 Cryptographic strength

Cryptographic strength of keys shall be at least 112 bits.

The strength of a key that is encrypting other keys should be equal to or greater than the key being encrypted. A key ever encrypted by a weaker key inherits the cryptographic strength of the weakest key with which it was ever encrypted. Observe that this weakening propagates down through a key hierarchy. For example, encrypting an AES 128 key with a triple-length TDEA key reduces the AES key's cryptographic key strength to that of the TDEA key, which is 112 bits. See ISO/TR 14742 for more details on cryptographic strength where various algorithms are used. A means to track the effective cryptographic strength of a key shall exist.

If TDEA is used for a new implementation, such implementation should utilize unique keys per transaction.

4.1.6 Key locations

A symmetric or private key shall exist in the minimum number of locations as is necessary for operations. The fewer the instances of the key, the stronger the claim of non-repudiation, as there is a lower probability of compromise.

4.1.7 Single-purpose key usage

4.1.7.1 Single-purpose symmetric key usage

An HSM MFK shall not be used as a key between organizations or logical systems for transport of other keys. A logical system would include a single system with multiple HSMs

Other than HSM master keys, each symmetric key shall have an intended purpose designation (IPD). Such IPDs shall be used by the SCD to restrict the use of the key to no more than one item from each of the following four classifications:

a) algorithm with which the key is to be used:

   1) AES;

   2) TDEA;

   3) HMAC;

b) cryptographic function:

   1) encryption/decryption;

2) authenticated encryption;

3) derivation;

4) message authentication;

5) key wrapping;

c) data element or type of information involved:

1) account holder authentication element (e.g. PIN) ;

2) keys;

3) other data element not listed previously;

d) functional direction or mode of use:

1) encrypt/generate/wrap;

2) decrypt/verify/unwrap;

3) both.

Where the IPD is used to indicate functional direction, the same key may have more than one IPD, depending on its environment. IPD configurations enforceable by the SCD shall be cryptographically bound to the key.

Keys electronically transmitted from the generating SCD shall have the IPD identified and bound to the key, which can be accomplished by setting attributes in a key block.

Keys shall be given the most restrictive IPD consistent with the function the key will be supporting in the implementation (e.g. encryption only or decryption only as opposed to both encryption and decryption). There are keys that are by nature bidirectional, such as a master key within an SCD (e.g. master file key).

4.1.7.2 Single-purpose asymmetric key usage

Each asymmetric key pair shall have an IPD. Such IPDs shall be used by the SCD to restrict the use of the key to only one item from each of the three following classifications:

a) algorithm with which the key is used;

b) cryptographic function (except for self-signed CSR):

1) encryption;

2) decryption;

3) signing;

4) verifying;

5) key agreement;

c) Data element or type of information involved, including:

1) keys related to confirming SCD legitimacy and/or functionality (e.g. manufacturer keys for verification of legitimate parts, code)

2) keys related to SKDAT;

3) PINs;

4) data.

The IPD should be used by the SCD to restrict the use of the key for communicating pairs to either:

a) terminal-to-host; or

b) host-to-host.

Each key shall be given a single IPD. IPD configurations enforceable by the SCD shall be cryptographically bound to the key

Keys electronically transmitted from the generating SCD shall have the IPD identified and bound to the key, which can be accomplished by setting attributes in a key block.

4.1.7.3 Additional intended purpose designations (AIPD)

In addition to the segmentation required previously, AIPDs may also be used to provide for segmentation classifications that are enforced by policy and procedure, rather than by the SCD.

An AIPD should be used to restrict the use of the key for communicating pairs to either:

a) terminal-to-host; or

b) host-to-host.

Examples of other AIPD classifications include, but are not limited to:

— designation as production or non-production;

— additional data element segmentation (e.g. separating PAN from other data types or segmenting an implementation-specific data element);

— BDK segmentation; for entities processing or injecting DUKPT keys or keys for other key-derivation methodologies, this might be done to segment the risk associated with a BDK compromise.

4.2 Secure cryptographic device

4.2.1 General requirements

Keys are managed within devices such as PIN entry devices, Hardware Security Modules and key loading devices. These devices shall meet the requirements of an SCD as described in the ISO 13491 series.

Some key management activities necessitate the use of non-SCD devices where their form factors have inherent limitations preventing them from meeting some SCD security requirements, especially tamper-responsiveness. These limitations have associated security risks which shall be addressed by restricted usage and additional controls. These devices are hardware management devices (HMD). Examples of HMDs include, but are not limited to:

a) smart cards used for component or share transport or storage;

b) smart cards containing public or private key pair(s) used to facilitate management of HSMs;

c) devices used to authorize or enable key management functions.

HMDs shall only be used in cases where the compromise of a single HMD would not compromise keys or secrets not held within that HMD.

Where there are requirements in this document that apply to an SCD, such requirements shall also apply to an HMD, unless specifically excluded.

If multiple cleartext components or shares sufficient to form a key are stored or transported within a single SCD, it is equivalent to having a clear key in the SCD; therefore, the SCD shall meet tamper responsive requirements in ISO 13491-1. The SCD shall enforce management of the components or shares in such a way as to ensure split knowledge of the key in accordance with 4.1.2.

When an SCD is used to generate a key or key component, the SCD shall use a random or pseudo-random process such that certain values are not statistically more probable than others from the set of all possible values, in accordance with ISO/IEC 18031.

When a key is entered as components, the SCD shall combine all the components using an XOR operation. Figure 1 is an example of the XOR process used to combine key components.

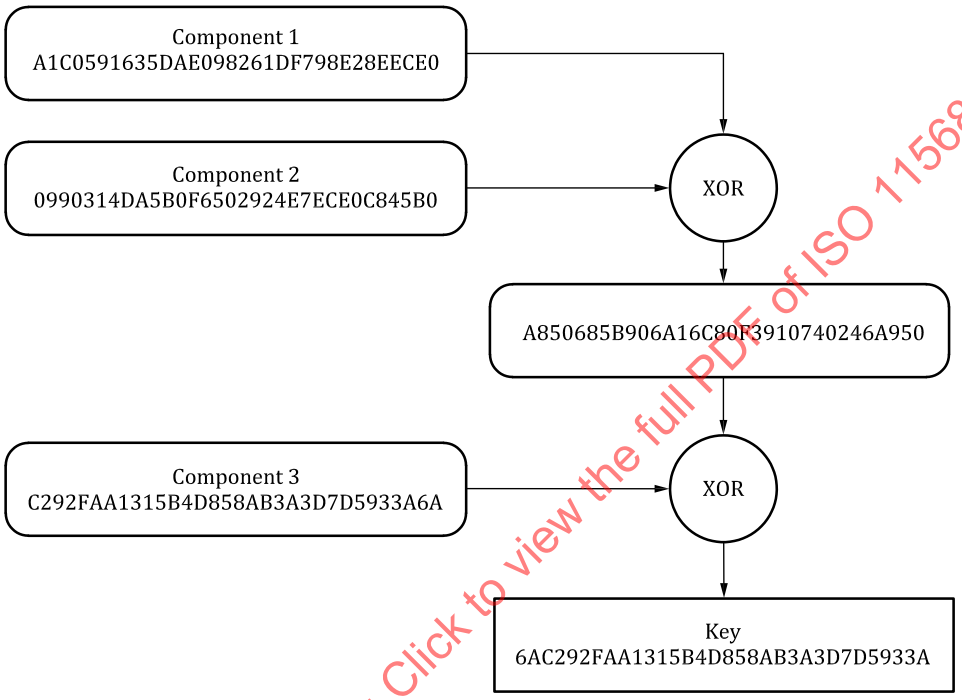


Figure 1 — Example of XOR function to combine key components

When an SCD is used to produce key shares, the SCD shall use an *m*-of-*n* secret-sharing scheme which is perfectly information-theoretically confidential and defined in ISO/IEC 19592-2.

The management and control of an SCD shall conform to the ISO 13491 series.

An HMD or an SCD that does not have a tamper responsive mechanism enabled shall be stored in an environment that meets at least the criteria of a minimally controlled environment as identified in the ISO 13491 series, with additional controls providing reasonable assurance that any unauthorized access to the device can be detected. Because the device cannot respond to an attack, the additional controls minimize the probability of unauthorized modifications.

An SCD is decommissioned when it is removed from service for repair, ownership is transferred to another organization or the SCD is to be destroyed. When an SCD is decommissioned, the financial keys shall be erased in accordance with ISO 13491-1. The keys should be erased as close to the time of removal from service as practical.

When an SCD is lost or stolen, all symmetric or private keys contained in that device should be considered compromised and the investigation outlined in 4.17 shall be initiated.

4.2.2 Additional SCD requirements for devices used in SKDAT

A KRD shall have mechanisms to prevent or detect middleperson attacks during the execution of SKDAT. A KRD shall have mechanisms that provide secure processing for authentication of the interfacing KDH.

A KDH shall have mechanisms to prevent or detect middleperson attacks during the execution of SKDAT and the ability to ensure communication with only an intended KRD. A KDH shall have mechanisms that provide secure processing for mutual authentication of the interfacing KRD.

A KRD may interface with more than on KDH for different key types (e.g. one for PIN keys and one for PAN encryption keys offered by a point-to-point encryption solution provider).

The first time a KRD interfaces with a KDH for a given key type (e.g. delivery of PIN-related keys), such mechanisms might only be sufficient to identify the KDH as having been one of a group of KDHs authorized by the device manufacturer or CA for that purpose. Once the KRD is authenticated to the intended KDH for a given key type, mechanisms shall exist to prevent communication with any unintended KDH for that same purpose.

4.3 Additional CA requirements

Where a CA is used for purposes of verifying the identity of participants in a key exchange protocol, then for certificate issuance the CA shall have processes in place to validate the identity of the certificate requester before issuing a digital certificate for the requester's associated public key.

The CA shall be implemented in accordance with the requirements specified in ISO 21188.

The CA shall document a certificate policy (CP) as specified in RFC 3647.

The CA shall document a certification practice statement (CPS) in accordance with RFC 3647 and the CA shall operate in accordance with its CPS.

In the event of a compromise of the CA system's cryptographic keys, the CA shall have procedures in place to revoke certificates and notify affected parties.

The CA system shall operate in an environment that meets the criteria of a secure environment as described in the ISO 13491 series, which provides secure processing to ensure the integrity of the public key infrastructure environment.

The CA system root private key shall only be operated to issue, revoke and suspend subordinate CA certificates. The CA system root private key shall be operated in a secure environment as defined in the ISO 13491 series. The CA system root private key shall be operated offline, i.e. in an "air-gapped" closed system. Access to the CA system root private key shall only be under controlled conditions using principles of dual control, observed by an independent witness. Audit or procedure records of each CA system root private key access shall be maintained such that all activities performed can be identified from the records alone.

All CA functions, including any registration functions, shall be executed on hardware that is physically separate from the key distribution host hardware.

4.4 Additional RA requirements

The Registration Authority (RA) is the function that is responsible for identification and authentication of the certificate subject (the entity identified in the certificate), but as it does not perform CA functions, it does not sign or issue certificates. An RA might assist in the certificate application process or revocation process or both. The RA does not need to be a separate body, but may be part of the CA.

The CA shall verify or require that the RA verifies the credentials presented by a certificate subject as evidence of identity or authority to perform a specific role in accordance with the certificate policy. The RA shall verify the certificate subject's possession of the associated private key through the use of a digitally signed certificate request pursuant to PKCS #10 or another cryptographically equivalent demonstration.

The RA shall also perform validation that the requestor is authorized by doing at least one of the following:

— confirmation by telephone, confirmatory postal mail and/or comparable procedure to the certificate subject to confirm that the organization has authorized the certificate application, confirmation of the employment of the representative submitting the certificate application on behalf of the certificate applicant and confirmation of the authority of the representative to act on behalf of the certificate applicant;

— confirmation by telephone, confirmatory postal mail and/or comparable procedure to the certificate subject's representative to confirm that the person named as representative has submitted the certificate application;

— confirmation that the requestor is authorized based on previously established trusted communication between the requestor and the RA (e.g. the SCD is directly connected to the certificate issuing CA within the manufacturer or repair zone or sending the public key of a KDH to the CA for signing over an authenticated communication channel).

The RA registration process to validate the certificate subject (SCD) may include, but is not limited to:

— checking the certificate subject's credentials evidencing claimed identity;

— verifying that identifying data provided by the certificate subject is valid;

— verifying that the identifying data pertain to the certificate subject;

— verifying that the certificate subject is entitled to obtain a public key certificate under the certificate policy;

— recording the binding of the certificate subject's identification data to the certificate subject's public key;

— checking the certificate request for errors or omissions in accordance with the certificate policy;

— direct inspection of the SCD;

— verifying a previously installed certificate;

— verifying that the serial number is on a validated list.

4.5 Key blocks

4.5.1 Overview of key blocks

When in storage or transmission, symmetric and private asymmetric keys shall be protected against misuse, substitution and modification through the use of key blocks. Public keys may be stored in the same general format of a key block without using encryption. This requirement applies to all symmetric algorithms and key sizes.

The key block protects the key and confidential attributes from unauthorized disclosure (using encryption) and protects all data in the key block from modification (using integrity protection).

Figure 2 is a high-level illustration of the structure a key block may have but is not intended to be prescriptive.

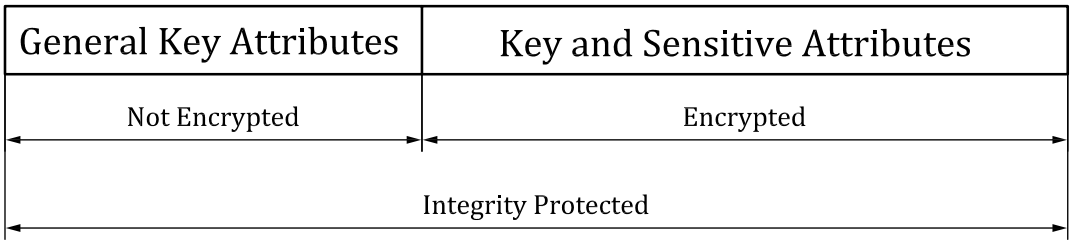


Figure 2 — Key block overview example

Except in the case of public keys, the key and its confidential attributes shall be encrypted. If the secrecy of the attribute is important to security of the key and key block, it shall be considered a confidential attribute.

The key and its attributes in the key block shall have integrity protection such that they cannot be modified without detection.

The key block shall contain the entire encrypted key (or component or share) and any attributes that are required according to this document. In addition, either the encrypted or the unencrypted portion of the key block may have additional data that is outside the scope of this document.

The process to create the key block is "key wrapping". ISO 20038 is an example of using an AES key to create a key block that meets the requirements in this subclause. ANSI X9.143 is an example of using a TDEA key to create a key block that meets the requirements in this subclause.

4.5.2 Key attributes

The key attributes define how the key can be used and other pertinent information. These are represented by values known as key tags, which are distinct constants (e.g. as defined in ISO 20038 for AES and ANSI X9.143 for TDEA). At a minimum, the attributes shall include the following:

a) One or more attributes as set by the IPD that define the operations for which the key can be used. Examples of attributes that define permitted operations include ones that specify that the key can be used only for PIN block encryption, only for encrypting other keys or only for computing Message Authentication Codes (MACs). Other attributes can define sub-categories, for example specifying whether a MAC key can be used to generate a MAC, to verify a MAC or both.

b) One or more attributes that define the cryptographic algorithm and mode for which the key can be used. These attributes are intended to prevent the misuse of a key using a different cryptographic algorithm or mode that could facilitate an attack to determine the value of the key.

Security requirements for key attributes are specified in 4.5.1.

4.5.3 Integrity of the key block

Before the key in the key block is used, the SCD that will use it shall verify the integrity of the key block with a cryptographic method that uses a key which is at least equal in strength to the key being protected. The key in the key block shall only be used if the integrity check passes.

4.5.4 Key and sensitive attributes field

4.5.4.1 Content of the field

At a minimum, the key and sensitive attributes field of the key block shall contain the value of the key but may also contain other data. For example:

a) Padding that has been used to extend the length of the cleartext key to some desired length before it is encrypted. One purpose of this is to pad a key to the block size of the algorithm used to encrypt

it. Another purpose is to hide the true length of the cleartext key, for example by padding so that all keys have the same encrypted length.

b) Information indicating the length and location of the key to assist in parsing.

c) Any sensitive attributes of that key that need to be kept secret.

4.5.4.2 Encryption of the key and sensitive attributes field

Electronic Code Book mode shall not be used to encrypt this field if the field is longer than one block.

The key and sensitive attributes field encrypted with AES shall use one of the methods identified in ISO 20038 or ISO/IEC 19772.

Methods using TDEA keys to encrypt the key and sensitive attributes field include, but are not limited to:

— the TDKW method from ANSI X9.102;

— methods defined in ANSI X9.143.

For further guidance, see ISO/TR 14742.

4.6 Key creation

4.6.1 Symmetric key creation

4.6.1.1 Random key generation

Keys shall be generated by using a random or pseudo-random process that ensures that no key generated is more probable than any other key. The strength of the random number generator shall be equal to or greater than the strength of the key it is generating. For further guidance on random number generation, see ISO/IEC 18031.

Keys shall be generated within an SCD.

4.6.1.2 Key derivation

Key derivation is a technique by which a (potentially large) number of keys are created ("derived") from a single secret key and non-secret variable data. The original single key is called a "derivation key" or "base key" and each key generated from it is called a "derived key". Key derivation can be used with a variety of key management methodologies and algorithms. For example, DUKPT allows for the derivation of a large number of statistically unique keys from a single key, eliminating the need to store each subsequent derived key. This is useful for an entity that drives a large number of devices, where each device is required to have unique keys.

Unique key per transaction (UKPT) is an example of a key derivation method used between two hosts. UKPT is a key management method to generate a session key based on a base key used by two communication partners exchanging a random value or counter to derive a unique key valid for the transmission of exactly one message or message exchange. It is up to the two organizations involved to determine the information that will be used to derive the unique key(s) for the transaction. See ISO/IEC 11770-6 for guidance.

It shall be infeasible to recover the derivation key from any set of derived keys. The procedure may be used iteratively, as a key generated from one initial key may subsequently be used as a derivation key to generate others. The derivation function shall conform to ISO/IEC 11770-6.

4.6.1.3 Key calculation (variants)

Applying a variant is a reversible technique for obtaining a set of keys from a single key, with each resulting key having a different key type. This technique provides key separation while eliminating the need to manage a separate, unrelated key of each required type. Each variant key is calculated from the original key and one constant from a set of non-secret constants using a reversible process. An example of such a process is the modulo-2 addition (i.e. XOR) of the key and a non-secret value. The disclosure of one key calculated in this manner discloses the original key and all other keys calculated from it.

A variant of a key shall be used only for key separation based on usage and the variant(s) of a key shall only exist in the same SCD that contains the original key.

Where variants are used with double-length TDEA keys, and one variant key is used to encrypt data that will also be available in the clear (e.g. PANs), the original key and associated variant keys shall be replaced regularly, in accordance with 4.13.

If a key is compromised, then all variants of the key are also compromised. If there are multiple variants of a key, and any one of those variant keys is compromised, then all variants and the original key are also compromised.

4.6.2 Asymmetric key creation

4.6.2.1 Introduction

The asymmetric key pair generation is the process by which a new pair of keys composed of a private key and the related public key is generated for use in a specific asymmetric cryptosystem. The two keys of an asymmetric key pair are mathematically related as defined by the design of the particular asymmetric cryptosystem. The relationship is such that it is computationally infeasible to determine the private key from the public key.

4.6.2.2 Requirements

The following requirements in this subclause apply regardless of the organization performing the generation activities.

Asymmetric key pairs shall be generated:

a) only within an SCD as specified in 4.2;

b) in such a way that guarantees the secrecy of the private key and the integrity of the public key;

c) in accordance with relevant reference(s) (the ISO/IEC 11770 series for RSA and Diffie-Hellman, ANSI X9.63 for Elliptic Curve and ISO/IEC 18032 for Prime Number Generator);

d) using an ISO-approved method for random number generation (see ISO/IEC 18031).

If the key pair is generated by a system that will not use it, the following applies to the transfer or transport of the key pair:

— The private key shall not be available in human-comprehensible form, i.e. it shall be encrypted.

— The private key of the key pair and all related secret seed elements shall be deleted from the sending SCD immediately after the transfer has been confirmed.

— The integrity and confidentiality of the private key shall be ensured throughout the transfer process.

— The public key shall only exist in one of the forms in 4.1.3.2.2 and should be delivered within a certificate if a CA is used.

Unique asymmetric key pairs shall be generated for each function (e.g. encryption versus signature) for each entity.

Asymmetric key pairs should have a replacement date to establish their life cycle.

4.7 Key component and key share creation

Both symmetric and asymmetric keys can exist as an *m* of *n* key share scheme with a minimum of two shares. Only a symmetric key can exist as two or more full-length components. The processes to create components or shares shall be performed using the principles of dual control and split knowledge. Components can be generated individually and combined to form a key or created from an existing key. The process to manage a key as shares requires the generation of the key and subsequent creation of the shares.

Key components shall be created by using a random or pseudo-random process such that no value is statistically more probable than any other from the set of all possible values. See the ISO/IEC 18033 series.

Key shares shall be created using a secret sharing method defined in ISO/IEC 19592-2, which is identified as perfectly information-theoretically confidential (also known as perfectly information-theoretically secure).

Except by chance, an individual key component shall only be used in the formation of a single key (i.e. the reuse of a component to form more than one key is not permitted).

Key shares and components for a new or existing key shall be created within an SCD.

If component creation or fragmentation into shares produces clear, human readable values, the process shall occur in at least a controlled environment as defined in the ISO 13491 series.

During the creation process, cleartext components shall only be displayed on or printed by an SCD. Cleartext components produced during creation and displayed on the SCD may be recorded on paper by the assigned key custodian.

If components are to be stored or transported, TEA bags shall be used. TEA bags are not required if components are created, loaded and destroyed in the same activity and location.

4.8 Check values

4.8.1 Introduction

A check value (CV) is used to verify that a value (e.g. a key, key component or key share) is as expected without disclosing the value itself. At any time following initial generation of the CV, repeating the same function with the same value will produce the same results. If the resulting CV is identical to the initial CV, it is assumed that the value is unchanged.

CV verification can be used to reasonably establish that one or more of the following conditions have been met:

— A value has been correctly entered into a cryptographic device.

— A value has been correctly received over a communications channel.

— A value has not been altered or substituted.

For example, an organization receiving a key, component or share from another organization might wish to have a method to confirm that the values received have been loaded correctly. CVs generated on keys are called key check values (KCVs). CVs generated on key components are called component check values (CCVs).

NOTE Because of the limited size of the CV, in a large set of cleartext values (keys, components, shares), different values can have the same CV (collisions).

If a KCV is used to verify keys and the calculated KCV does not match the expected KCV, the key shall not be used until an investigation is performed to determine the cause of the mismatch.

4.8.2 Symmetric key check value calculation

A check value should be calculated with a key or component at the time of generation. When a check value is calculated for a key or key component, it shall be calculated by a cryptographic process such that all portions of the key or key component are involved in calculating the check value. It is a significant security risk to calculate check values on portions of the key individually (e.g. a separate check value for the right and left halves of a double-length TDEA key). Thus, in the description of this algorithm, the term "key" always refers to the entire key (e.g. all 112 bits of a double-length TDEA key and all 192 bits of a triple-length TDEA key).

Annex A details symmetric key check value calculation. When calculating CVs for AES keys, A.3 shall be used. When calculating CVs for TDEA keys, A.2 or A.3 shall be used. When generating a key check value for an HMAC key, it shall be computed according to A.4.

When displaying or recording the component check value or key check value computed according to A.2, no more than the six hexadecimal digits of the cipher text shall be used.

When displaying or recording the component check value or key check value computed according to A.3, no more than the ten hexadecimal digits of the cipher text shall be used.

4.8.3 Asymmetric key check value calculation

For asymmetric keys, the reference KCV may be computed as the hash of the public key, and optionally associated data, using an algorithm defined in ISO/IEC 10118.

For public keys, as long as the KCV is distributed via an integrity-assured channel, the public key can be distributed via a non-secure channel. Prior to installing the public key for use, the user shall validate it by recomputing the KCV and comparing it with the reference KCV.

It shall be infeasible to modify or substitute the reference KCV and public key (and associated data) in such a way that the recomputed KCV of the modified or substituted public key (and associated data) equals the modified or substituted reference KCV. This can be achieved by either of the following:

— ensuring the integrity of the KCV (see ISO 16609);

— separately storing or distributing the reference KCV in such a way that modification or substitution of the reference KCV cannot be coordinated with modification or substitution of the public key (and associated data) (e.g. dual controls).

4.9 Key distribution

4.9.1 Symmetric key distribution

4.9.1.1 Introduction

The distribution process for an encrypted key shall protect against the use of a key that has been substituted or modified in an unauthorized manner.

Compliant implementation of requirements for secure management of symmetric keys requires (among other things) unique key relationships and strict enforcement of dual control and split knowledge processes when handling cleartext keying material. This includes keys deployed to remote devices or established between communicating pairs. Historically, compliant implementation of key distribution has been a manually performed, physically on-site process that is difficult to manage, costly and/or non-existent (i.e. not compliant). An automated rather than manual method of distributing symmetric keys can address these issues and result in improved security of the financial services environment.

4.9.1.2 Encrypted with symmetric key

Keys, components, and shares encrypted with symmetric keys and contained in key blocks in accordance with 4.1.3.1 can be transmitted through unprotected communication channels.

4.9.1.3 Encrypted with asymmetric key

4.9.1.3.1 General

The use of public key cryptography and associated asymmetric key algorithms is one solution for automated remote symmetric key distribution, and therefore included in this document. This solution is referred to as symmetric key distribution using asymmetric techniques (SKDAT).

The functional roles for SKDAT are described in Annex K.

For SKDAT, two types of symmetric keys can be involved. The first is the symmetric key (e.g. key encrypting key, PIN encrypting key, DUKPT initial key) to be installed into the KRD and corresponding Acquirer host (if applicable) or to be distributed between two hosts. The second is a symmetric key that functions as a key encrypting key for purposes of encrypting a private key of a public–private key pair for storage in a database or during transport. The first type is the object of protection, as covered in 4.9.1.2; the second provides the protection and is subject to all the requirements of other symmetric keys. A SKDAT implementation shall include a method to bind the symmetric key being distributed to a particular cryptographic algorithm.

For automated SKDAT, examples of acceptable methods of binding the symmetric algorithm to the distributed key include but are not limited to:

— using cryptographically bound mechanisms, for example labels, to indicate the appropriate algorithm and usage;

— including the symmetric algorithm name or other unique identifier in the key derivation function of a key agreement scheme.

Keys, components and shares encrypted with asymmetric keys as described in 4.1.3.1 can be transmitted through unprotected communication channels.

At a high level, there are three phases of SKDAT in typical protocols using a CA. The following description is based on a PIN entry device (PED) as the KRD, but the concepts can apply to any KRD.

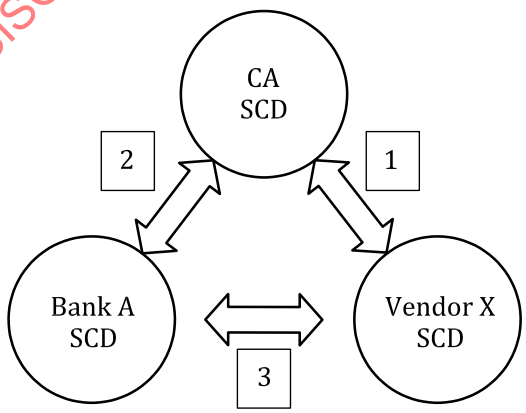


Figure 3 — Automated SKDAT phases

Phase 1 – KRD initialization: During the manufacturing phase of the KRD (PED), when the device is initialized (or at a subsequent re-initialization following repair), the KRD vendor will inject data such that the KRD can be reliably distinguished from a counterfeit device. This data typically consists of a certificate, which corresponds to the private key of the KRD. The KRD's key pair may be sent to the

KRD encrypted with a transport key or the KRD may self-generate the key pair and export a certificate signing request (CSR) to the CA and the CA responds with the certificate.

The KRD vendor will also inject the device with a CA certificate such that the KRD can validate KDH certificates issued by the same CA.

Phase 2 – KDH initialization: Prior to implementing SKDAT, the same CA certificate that was provided to the KRD is provided to the KDH. Also, similar to what happened in Phase 1 for the KRD, a KDH-specific key pair is generated by or installed (encrypted) into the HSM; the public key of that key pair is signed by the CA and the signed public key is installed in the KDH (host system, not in the SCD).

Phase 3 – Key exchange: When the KRD interfaces with a KDH, the KRD and KDH will exchange certificates. Each will use the CA certificate to verify the authenticity of the received certificate.

4.9.1.3.2 Phase 1 requirements

In Phase 1, the KRD public key that has not yet been signed shall be protected against substitution or modification. Sufficient controls shall exist to ensure that it is infeasible for a counterfeit device to be initialized such that it cannot be reliably distinguished from a genuine device. The CA shall have mechanisms to ensure that the CSR is received from a valid device. One method of achieving this is to contain the operations where the unsigned public key is presented to the CA for signature. If this method is used, the activity to sign the public key shall occur within a secure environment as defined in the ISO 13491 series.

Where the root level CA is used to sign a subordinate CA that issues certificates to a KDH or KRD, the authenticity and integrity of the root CA certificate installed with the signed sub-CA public key shall be verified prior to installation in the secure environment. The integrity of the CA certificate shall be maintained prior to loading into the KRD. Once the CA certificate is loaded into the KRD, it shall be protected against unauthorized modification, substitution or replay.

The request to generate or sign the KRD certificate shall include sufficient data for the CA to validate the authenticity of the request. The request channel to the CA shall be protected against unauthorized substitution, modification and replay. All requests to the CA to generate or sign certificates shall be logged by the requesting entity, including any requests that fail. The CA shall log all requests to generate or sign KRD certificates, retaining all element in the CSR, and enough information about the event to facilitate a forensic investigation.

4.9.1.3.3 Phase 2 requirements

The CA shall validate the KDH CSR in accordance with the registration authority requirements as defined in 4.4. The KDH certificate request to the CA should be sent through an integrity protected channel.

The CA public key shall be distributed as per the appropriate subclause of ISO 21188.

The integrity of the CA certificate shall be verified prior to use by the KDH HSM. Once the CA certificate is loaded into the KDH, it shall be protected against unauthorized modification, substitution or replay.

Signing of the KDH public key shall be performed in a secure environment as defined in the ISO 13491 series.

Procedural controls shall ensure that only authorized signed KDH public key(s) are installed for use.

4.9.1.3.4 Phase 3 requirements

In Phase 3, prior to transport of a symmetric key to the KRD, the KDH shall verify that this is the intended KRD (i.e. the KDH has prior knowledge of the identity of the KRD and does not rely only on the validity of the KRD certificate).

The KRD shall validate the legitimacy of the KDH prior to accepting the symmetric key and installing it for future use.

See Annex C for an overview of asymmetric distribution techniques.

4.9.1.4 Cleartext key distribution

4.9.1.4.1 General

Cleartext symmetric keys are distributed as components or shares of the key, or by using an SCD to transport the entire cleartext key.

Regardless of the key distribution method used to transport cleartext components or shares, it shall not be possible for any person to ascertain any of the bits of the final key during the distribution process.

In some cases, it is expedient to have authorized individuals carry (i.e. convey) components or shares of the key from one place to another (e.g. from the point of storage to the point of loading). Requirements for this type of situation are covered in 4.9.1.3.1. Subclause 4.9.1.4.3 and Annex B describe the requirements that apply when the transportation of the component(s) or share(s) is not under the physical control of authorized persons at all times (e.g. when mailing).

4.9.1.4.2 Personal conveyance of cleartext components or shares

When conveying keys, each cleartext component or share not in an SCD shall be in the physical control of only the authorized Custodian and should be in the possession of the authorized Custodian only for the minimum practical time required to transport the component or share, enter it into an SCD (if applicable) and then either securely store or destroy it.

The authorized custodian physically carrying a component or share in printed form shall ensure that the component or share value is not visible to any unauthorized individual during conveyance.

4.9.1.4.3 Transporting cleartext components or shares in an SCD

Cleartext key components or shares may be transported in one or more SCDs. This subclause covers the transport of a single component or share of one or more keys using an SCD (e.g. a single SCD containing "Custodian 1" components for one or more keys). Transporting multiple components or shares of a given key within a single SCD shall be considered equal to transporting a cleartext key within the SCD and is covered in 4.2.

The management of a component or share (including the SCD in which the component or share resides) shall ensure dual control and split knowledge of the resulting key.

For cleartext components or shares transported within an SCD, that SCD shall meet the following requirements in addition to those in 4.2:

a) have logical controls to limit access to authorized individual(s);

b) have logical controls that include protections against attacks (e.g. repeated failed login attempts);

c) support authentication controls that render the device or user account unusable and/or remove the keys when the number of invalid authentication attempts exceeds a predefined limit.

The process to transport the SCD shall conform to Annex B.

4.9.1.4.4 Transporting cleartext components or shares in printed form or on transport media

Transport media refers to any device that is used to contain a component or share that does not meet the criteria of an SCD (as defined in 4.2), such as an HMD.

The sender and receiver have equal responsibility for the secrecy of the cleartext components or shares that are transported in printed form or on transport media. Both parties shall have procedures to ensure the secure handling of the components or shares throughout the process in accordance with the principles of dual control and split knowledge as described in 4.1.2.

The process to transport key components or shares that are in printed form or contained in transport media shall conform to Annex B.

4.9.2 SKDAT asymmetric key distribution

4.9.2.1 Asymmetric key pair transfer

The asymmetric key pair transfer is the process by which the key pair and the certificate of the public key are conveyed to the owner of the key pair. This process occurs when the owner does not have the capacity to generate their key pair.

The owner shall be authenticated prior to being given their key pair.

The techniques used for public key distribution are described in 4.9.2.2 and plaintext private key distribution is covered in 4.9.2.3.

When the key pair is not generated by the key owner, prior to the distribution of the public key the correct transfer of the key pair should be verified.

4.9.2.2 Public key distribution

The public key of an asymmetric key pair needs to be distributed to, and stored by, one or more users for subsequent use as an encryption key and/or signature verification key, or for use in a key agreement mechanism. Public key distribution is the process by which a public key is conveyed to the party intended to use it. Although this key need not be protected from disclosure, during distribution to authorized recipients or during storage in a key database, it is necessary to ensure the authenticity of the public key.

Mechanisms for the distribution of asymmetric public keys are described in ISO/IEC 11770-3.

Any distribution method (manual or automated) shall ensure the integrity and authenticity of the public key. The substitution of a public key during distribution shall be prevented. This can be achieved by maintaining the public key in the forms described in 4.1.3.2.2.

An independent communication shall be used to verify that the identification of the key and its owner are correct and authorized. This might require confirmation obtained via a different channel from the one whereby the information was originally obtained.

One or more of the following techniques shall be used to ensure public key integrity:

— sign the public key and associated data using a digital signature system, thereby creating a public key certificate in accordance with 4.3 and 4.4;

— create a MAC for the public key and associated data, using an algorithm defined by ISO 16609 and a key used only for this purpose;

— store the public key in an SCD (see the ISO 13491 series);

— distribute the public key over a channel that protects both the integrity and authenticity of the public key (e.g. VPN);

— distribute the public key over an unprotected channel and distribute a KCV of the public key and associated data over a channel that protects both the integrity and authenticity of the KCV; asymmetric KCVs are described in 4.8.3;

— use authenticated encryption;

— transport within a CSR.

4.9.2.3 Private key distribution

If it is necessary to output the private key from the generating SCD (e.g. for transfer to another SCD where it is to be used or for backup purposes), it shall be protected from compromise by at least one of the following techniques:

— encipherment with another cryptographic key;

— if non-encrypted and outside an SCD, as key shares using an acceptable key segmentation algorithm (see ISO/IEC 19592-2).

One or more of the following techniques shall be used to ensure private key integrity:

— Create a MAC for the private key and associated data, using an algorithm defined by ISO 16609 and a key used only for this purpose.

— Use authenticated encryption.

— Use key blocks as defined in ISO 20038 for AES or ANSI X9.143 for TDEA.

— Store as integrity-protected key shares.

— Verify that the private key and the authenticated public key form a valid key pair by first applying a transformation on arbitrary data using one of the keys, then applying the complementary transformation using the other key and then confirming the result against the expected result. This operation shall be wholly conducted within an SCD and all intermediate and final results of the transformation destroyed.

Key shares that will form a private key shall be distributed in conformity with the requirements for symmetric key shares. See 4.9.

The process of transferring encrypted private keys shall protect against key substitution and modification.

The key's identifier and related data should be transferred together with the private key.

Where the SCD that will use the private key is not the SCD that creates it, the private key shall not be recorded or retained by the source SCD.

4.10 Key loading

4.10.1 General

The loading of keys into an SCD shall be performed using one of following techniques:

— manual entry (i.e. typing cleartext components);

— electronic direct loading (e.g. using a KLD or an HMD);

— distribution over a network (i.e. SKDAT or master key or session key).

The permissible techniques to load keys into an SCD as a function of the different key forms are indicated in Table 2.

Table 2 — Permissible key distribution and loading techniques

Key forms	Techniques		
	Manual	Electronic	
		Direct	Network
Plaintext keys	Not permitted	Not permitted	Not permitted
Plaintext key components or shares	Permitted	Permitted	Not permitted
Encrypted keys	Permitted	Permitted	Permitted
Encrypted key components or shares	Permitted	Permitted	Permitted

Regardless of the key loading method used, it shall be infeasible for any person to ascertain all or part of any final key during the key loading process.

Keys shall be loaded into an SCD only when it can be ensured that the device has not been subject to prior tampering which might lead to the disclosure of keys or sensitive data.

Except for sending encrypted values to an SCD that is housed in at least a controlled environment as defined in the ISO 13491 series, the key loading process shall be performed according to the principles of dual control and split knowledge.

A key shall not be reloaded to a transaction-originating SCD, except by chance.

To facilitate the detection of errors in key loading, the check value for key components and keys should be verified.

Key loading activity shall be recorded in an automated and/or manual log that contains:

a) an identifier specific to the physical device of any hardware involved (e.g. serial number);

b) the custodian(s) involved;

c) a key identifier (e.g. a key name, KCV, BDK ID);

d) the key type;

e) the identifier of component(s) or share(s) loaded, if applicable (e.g. component 1, share B, check value);

f) the date and time of the activity.

4.10.2 Loading key components or shares

The integrity and authenticity of the physical mechanism protecting a cleartext key component or share shall be verified prior to installation into the target SCD. Physical mechanisms include but are not limited to:

— TEA bags;

— transport SCDs.

Cleartext key shares or components shall be loaded into an SCD only when it can be ensured that there is no active or passive tapping mechanism at the interface that might disclose the transferred shares.

During loading, each cleartext component not in an SCD shall be in the physical possession of only the authorized custodian and shall be in the hands of the authorized custodian only for the minimum practical time required to transport the component or share, load it into an SCD and either securely store or destroy it.

The authorized custodian physically handling a cleartext component or share in printed form shall ensure that the component or share value is not visible to any unauthorized individual during conveyance and entry.

The loading process shall not disclose any portion of a key share to an unauthorized person.

A key component or share shall be entered into a target SCD using one of the following methods:

— entered directly into an input mechanism that is integral to (i.e. within the SCD) the target SCD in such a way as to protect against monitoring (e.g. using the EPP of ATM, or the KLD used to load components or shares to an HSM);

— in encrypted form as outlined in 4.1.3;

— entered from a directly connected portable source SCD (i.e. HMD) that was used to transport the components of the key to be loaded into the target SCD as described in 4.9.1.3.2.

Once a cleartext component or share has been loaded, it shall either be destroyed immediately or transported securely for subsequent storage or destruction. Transportation of component(s) or shares(s) shall conform to 4.14.2. Whether destruction of key component(s) or share(s) is done immediately or subsequent to secure transport, it shall conform to 4.14.

4.11 Key utilization

4.11.1 General key utilization requirements

Physical and logical controls shall be implemented to prevent unauthorized key use.

Keys that are known (or where their cleartext value is available during any process) or have ever been protected by known keys shall not be used in production.

Any key created or protected in equipment that does not meet the requirements of 4.2 shall not be used in production.

Any keys used in a production system shall be handled in accordance with the requirements in this document.

Any symmetric or asymmetric private key used by a communicating pair (e.g. host-to-host or PED to host) shall be unique to that keying relationship (other than by chance). While a communicating pair could be a transaction-originating terminal and its host, this requirement is not meant to imply that a Base Derivation Key (BDK) has to be unique to a PED or a single merchant. A host is a single organization that may include multiple locations, systems and devices. Load balancing among HSMs within the same host is permitted.

Any key resident in a transaction-originating SCD shall exist only in that device and those facilities that are authorized to have the key. If the transaction-originating SCD has multiple cryptographic relationships, there shall be unique keys for each cryptographic relationship. For example, a PED that communicates with Acquirer 1 and Acquirer 2 would have different keys for each Acquirer. In cases where a PED communicates with an Acquirer with multiple locations, the same key can be used at each location.

Entities using DUKPT or other key management strategies using derivation methodologies should incorporate a segmentation strategy (e.g. by financial institution, injection vendor, market segment, processing platform, sales unit) in their environments to limit the impact of a compromise.

A key shall be used for one purpose only. Policies and procedures shall be in place to support and require an IPD for each key. The IPD may be a single designation or a set of separate designations that collectively determine the appropriate use of the key. The use of key blocks as described in 4.5 aids in the enforcement of the designations.

SCDs shall provide IPD enforcement for the required IPD classifications as applicable, based on functionality supported by the SCD.

An SCD shall have a means to ensure the separation of keys according to cryptographic function.

A key shall not be used to encrypt itself.

The authenticity and integrity of the public key shall be verified prior to each use or the public key shall be maintained in such a manner that the authenticity and integrity are ensured during operational use.

The key encryption key used for the archival process shall not be the same (except by chance) as any of the key encryption keys used to encrypt active keys.

4.11.2 Additional key utilization requirements for SKDAT

If the symmetric key is known or accessible during a test process, then that key shall not be distributed with production certificates used in SKDAT.

If the private key is known or accessible during a test process, then mechanisms shall exist to ensure that the key pair cannot be used in a production environment and the public key shall not be signed using a production CA signing key.

Private keys shall only be used at the location where they are intended to be used and shall only exist in the minimum number of locations necessary.

Private keys shall be used for only one of the following:

— decryption;

— creating digital signatures;

— key agreement.

Any such key may also be used for self-signing a certificate.

A public key's authenticity shall be verified and, if it cannot be authenticated, it shall not be used.

4.12 Key storage

4.12.1 Cleartext key component and share storage

Cleartext key components and shares include written form and those stored within an HMD, regardless of whether that component or share is encrypted within the HMD or similar device.

The storage of cleartext components and shares shall ensure that only the authorized custodians similarly entrusted with that component or share can obtain access to the component or share. Key components shall be stored in such a way that unauthorized access (even by an authorized custodian) has a high probability of being detected. Each key component or share should be stored under dual control, ensuring that each access is for an authorized activity.

Each component and share shall be stored in its own separate sealed TEA bag.

The chain of custody documentation of a key component or share shall be maintained for the life of the key.

Each time a TEA bag is placed into or removed from storage, there shall be an inspection for evidence of tampering and validation that the TEA bag identifier is as expected.

4.12.2 Public key storage

Protection against substitution of the public key during storage is essential. For example, the substitution of a public key used for encryption could result in a threat to data secrecy.

One means of protecting a public key against substitution is to implement the same techniques as for a private key. Another means is to store the public key in a certificate, allowing verification of the key's integrity and authenticity before use.

If unauthorized key substitution is known or suspected, the public key shall be updated with the correct public key. See also 4.17.

4.13 Key replacement

A cryptoperiod is the time span during which a specific key is authorized for use. Cryptoperiods serve to:

a) limit the information (related to a specific key) available for cryptanalysis;

b) limit exposure in the case of compromise of a single key;

c) limit the use of a particular technology to its estimated effective lifetime;

d) limit the time available for computationally intensive cryptanalytic attacks (in applications where long-term key protection is not required).

A key rotation policy shall exist that defines the cryptoperiod for keys owned by the organization. The cryptoperiod of a key shall be no longer than the least time deemed feasible to perform a dictionary or key exhaustion attack, see ISO/TR 14742 for guidance on usable key life. This time will depend upon the specific implementation and the technology available at the time of the attack. At or before the conclusion of the key's cryptoperiod, it shall be replaced. Considerations for the rotation schedule include (but are not limited to):

— key use;

— key type;

— key strength;

— amount of data being encrypted;

— key management history;

— key management methodology (see 5.1).

A key should be retired if it has been managed as more than two components or shares and clear history of a single component has not been maintained. Key compromised is covered in 4.17.

Public keys, when generated and issued for use, typically have an expiry date that determines the key pair life cycle. Public keys shall not be used beyond their expiry date.

An unauthorized attempt to enter a key or restore a replaced symmetric key (e.g. replay) shall be precluded or detected.

Key replacement shall be implemented by one of the following methods:

a) Repeating the appropriate key generation, distribution and loading procedure.

b) Transmitting to an SCD a working key encrypted under a key encryption key which has previously been placed in the device.

c) Generating a new transaction key within the device as the non-reversible transformation of a previous key.

   1) One possible objective is to obtain a unique key per transaction in an automatic way. The transaction key or the data needed to produce the transaction key exists only for the duration of a single transaction or the time between two consecutive transactions.

   2) When such a unique-key-per-transaction technique is used it shall not be possible to determine any previous transaction key from the information contained in an SCD after a transaction has been completed.

d) Implicitly distributing a new key by computing a new transaction key within the device as a function of a Current Key and transaction data.

If key replacement is being performed to prevent a dictionary attack on the encrypted data, any one of the four methods may be used.

If key replacement is being performed to limit the effect of a future physical compromise of the SCD, only methods a) or c) shall be used.

If key replacement is being performed because of known or suspected compromise, methods a) or b) shall be used.

In the case of key replacement, both the public and the private key of a key pair shall be replaced.

Keys that have been replaced shall not be restored or returned to active use.

4.14 Key destruction

4.14.1 General

Keys shall be destroyed when they are no longer operationally necessary. Keys can exist in multiple forms in multiple locations and at multiple organizations. The process of destroying a key shall involve all known forms of the key.

Paper-based keying materials shall be destroyed by crosscut shredding, burning or pulping. When material is burned, the residue should be reduced to white ash. Regardless of the method of destruction of paper-based keying material, the following shall apply:

— The entire piece of paper on which the component is printed or written shall be consumed by the process.

— The keying material shall be rendered unrecoverable.

Keying material stored on other media shall be destroyed so that it is impossible to recover by physical or electronic means.

Destruction of keys shall be accomplished under conditions of full accountability, with appropriate records retained for audit trail purposes.

SCDs that have been permanently removed from service shall have all symmetric cryptographic keys within them destroyed. In addition, if the SCD is a PED, all host public keys and identifying host certificates stored within that PED should be destroyed. In addition, if the SCD is an HSM, all public keys and identifying certificates of the communicating pairs involved in the exchange of the symmetric keys should be destroyed.

SCDs that have been taken out of service for repair or for destruction shall have all symmetric keys and asymmetric private keys within them destroyed.

4.14.2 Key destruction from an SCD

The method of destruction for keys inside an SCD shall be sufficient to ensure that it is infeasible to recover the keys by physical or electronic means.

All manually initiated destruction activities, including the purposeful initiation of a tamper state, shall be logged, with appropriate records retained for audit trail purposes.

4.14.3 Destruction of a key in cryptogram form

In order to destroy the cryptogram form of a key, the destruction process shall include one or more of the following:

— removing the key cryptogram(s) from the operational environment(s);

— destroying all forms of the key used to create the cryptogram.

Operational processes shall ensure that destroyed keys are not reintroduced.

4.14.4 Component and share destruction

Components or shares stored on non-paper media shall be destroyed so that it is infeasible to recover the component or share by physical or electronic means. The destruction shall be completed by the authorized component holder and be witnessed by another authorized individual. For guidance related to methods specific to media types, see ISO 9564-1:2017, Annex A.

Paper-based components or shares shall be destroyed by crosscut shredding, burning or pulping. The destruction shall be completed by the authorized component holder and be witnessed by another authorized individual. When material is shredded, all residue shall be reduced to pieces 5 mm by 5 mm or smaller. When material is burned, the residue shall be rendered unreadable, for example reduced to white ash.

4.15 Key backup

Key back up is storage of a copy for the purpose of reinstating a key that is accidentally destroyed, but the compromise of which is not suspected.

Keys maintained for backup shall be managed according to this document.

4.16 Key archiving

An archived key is an inactive key that is being saved in a secure manner for a non-operational purpose. Legal or regulatory requirements and forensic investigations are examples of instances that can require recovery of an archived key.

Examples of specific key types that might need to be archived are:

— a public key used for signature verification (it could be needed to validate signatures after its retirement from operational use);

— a key to decrypt data that has been encrypted.

NOTE For additional information related to archiving keys, see NIST SP 800-57-1.

Archived keys shall be managed in accordance with this document.

Archived keys shall not be restored into the operational environment.

An archived public key shall only be used to verify the legitimacy of transactions that occurred prior to archiving. After such verification, the instance of the key necessary to perform the verification shall be destroyed.

Archived keys shall be securely stored for the life of all data or keys encrypted under such keys.

A key shall be archived in such a way that the risk of exposure of keys that are still in operational use is not increased.

An archived key shall be retained for no longer than is necessary to meet applicable obligations.

A procedure shall be established for adequate access control to archived keys, components and shares.

Archived keying material shall be stored separately from operational keying material.

The key encryption key used for the archival process shall not (except by chance) be the same as any of the key encryption keys used to encrypt active keys.

4.17 Key compromise

A key compromise occurs when the confidentiality or integrity of a cryptographic key can no longer be depended upon because an unauthorized disclosure of the key or its components or shares might have occurred. There need not be any evidence of misuse, nor affirmative confirmation that the information actually was exposed, only an inability to demonstrate reasonable assurance that it remains secret.

An event or discovery that leads to potential distrust of the key is known as a suspected compromise.

Examples of a suspected compromise include (but are not limited to):

a) an SCD showing signs of or displaying an alert that indicates possible tampering;

b) missing SCDs in which keys were loaded;

c) inconsistent key activity logging or a violation of the security of a system such that an unauthorized disclosure of the key or information it protected might have occurred;

d) components not stored in sealed TEA bags such that it prevents the disclosure of the contents without noticeable damage to the packaging;

e) TEA bags or component storage containers showing signs of tampering attempts;

f) TEA bag records inconsistently maintained, gaps in the TEA bag history for any one component of the key or evidence that a TEA bag has been opened without authorization;

g) a published vulnerability that might impair security of the key (e.g. CVE, OWASP).

When a suspected compromise is detected, an analysis shall be initiated to identify if a compromise has occurred. The analysis shall be documented. The documentation shall include, at a minimum, the following information:

— a summary of the concern that initiated the analysis;

— copies of pertinent evidential matter;

— steps taken to ascertain if a compromise has occurred;

— a conclusion as to whether the key is compromised (i.e. was reasonable assurance of secrecy of the key demonstrated by the investigation);

— the rationale upon which the conclusion is based.

If one or more components or shares of a key are compromised and the result is that one person could have (or have had) knowledge of the resulting key, the key shall be considered compromised. For example, if the key exists as two components and a single component is lost or exposed, if the legitimate custodian of the remaining component were to discover it, he or she would then have had access to the entire key. If the key exists as more than two components or shares and a single component or share is lost or exposed, then the key is not necessarily considered compromised. Similarly, if one share of a

key is in question, as long as there is no condition in which a person has access to one fewer than the number of shares sufficient to form the key, then gaining access to that single share would not expose the entire key.

In a situation where a single person is not able to gain knowledge of the entire key, but one or more components or shares of a key have been lost, compromised or suspected compromised, the entire key should be replaced.

The use of a compromised key and any key protected by a compromised key shall be discontinued. If the compromised key is a derivation key, then use of any key derived from that key shall be discontinued. A key and its variants shall be replaced if either the original or a variant key is compromised. If a symmetric key is compromised, all asymmetric private keys encrypted under that symmetric key shall cease to be used and shall be replaced only after the compromised master key is securely replaced. Appropriate measures shall be in place to ensure the revocation of any affected digital certificates, if applicable.

A compromise analysis of a CA's private key shall include a determination of whether subordinate CAs and KDHs need to have new certificates issued using a new key and distributed to them or be notified to apply for new certificates. If a CA's or device manufacturer's private key is compromised, the CA or manufacturer shall cease issuing certificates signed by that key. All certificates relying on that key for its chain of trust shall be revoked.

The CA or manufacturer shall provide notification of the compromise to all organizations with which they have key management relationships and should make best efforts to communicate with other affected parties.

If a KDH private key is compromised, the KDH shall cease using the compromised key pair, generate a new key pair and have the public key signed, and distribute the new public key to all affected parties. The KDH shall also establish new symmetric keys with all affected parties.

The SKDAT implementation shall have a method to prevent delivering new keys to a KRD if the KRD's private key is known to be compromised.

The notification of a key compromise shall be communicated to all parties that could be relying on the secrecy of the compromised key.

Key destruction shall be in accordance with 4.14.

No compromised key shall be used in the process to create the replacement key.

5 Transaction key management techniques

5.1 General

Alongside describing the master key or transaction key and unique key per transaction, this subclause specifies the methods of transaction key management. Key management methods used individually or in combination shall be restricted to the following:

— master keys or transaction keys;

— host-to-host unique key per transaction (UKPT);

— derived unique key per transaction (DUKPT) as described in 5.3.

5.2 Method: master keys or transaction keys

This method utilizes a key hierarchy whereby a key encryption key (KEK) is considered to be at a higher level than the key that it encrypts. The simplest is a two-level hierarchy, whereby working keys are encrypted by KEKs which are themselves stored in a cryptographic device. In a three-level hierarchy, these KEKs are also managed in an encrypted form using a higher-level KEK (e.g. a master file key).

In this method, the master key is a key encrypting key (KEK) used for the distribution of new or replacement key encrypting keys, derivation keys and/or transaction keys. This method is also known as the master key/session key method. A KEK shall not be used to encrypt its replacement.

The highest-level KEK shall be either physically loaded (see 4.10) or remotely loaded using SKDAT. A key shall not be used to encrypt itself or its replacement (except as permitted in 4.11).

The effective and minimum strength of an encrypted key (e.g. transaction key) shall be determined according to 4.1.5. Since the master key is a key encrypting key, it should have cryptographic strength equal to or greater than the keys it encrypts.

Within a cryptographic relationship, if one organization is storing or transmitting the key shared between them using a key that is not of equal or greater strength, that organization shall notify the sharing party. The effective strength of a key shared between two organizations shall be equivalent to the lowest effective strength in either organization. For example, if two organizations share a 128-bit AES key, one organization encrypts the key under a 112-bit (double-length) TDEA master file key and the other organization encrypts the key under an 128-bit AES master file key, the strength of the shared key for both organizations is 112 bits, based on application of the logic in 4.1.5.

5.3 Derived unique key per transaction

5.3.1 General

This document includes AES DUKPT as a method to derive unique initial DUKPT keys and unique transaction key(s) from a single base key. Keys that can be derived include symmetric encryption or decryption keys, authentication keys and HMAC (keyed-hash Message Authentication Code) keys. AES DUKPT supports the derivation of AES-128, AES-192, AES-256 and double- or triple-length TDEA keys from AES-128, AES-192 and AES-256 initial keys.

Implementations using AES DUKPT as described in this document provide for the generation of unique transactions key(s) from an initial DUKPT key, in such a way that (i) the originating device does not preserve any information that could be used to derive the transaction key after the transaction has been completed and (ii) the Hardware Security Module at the receiving institution can efficiently derive the same transaction key(s) from a BDK with limited information stored in the receiving system. This is an update to the original DUKPT algorithm method based on TDEA. This update is based on AES and offers a number of other security improvements. Keys can be derived for use with either the AES or TDEA algorithm.

The AES implementation described in this document is recommended for new DUKPT implementations. For legacy purposes, the original TDEA DUKPT implementation as described in ANSI X9.24-1-2009, Annex A is included in this document as Annex I.

For AES DUKPT, pseudocode notations are given in Annex G and test vectors are given in Annex H.

DUKPT may be implemented using other approved 16-byte block ciphers in ISO/IEC 18033-3, but specifics are not included in this document.

5.3.2 DUKPT key management

The transaction-receiving SCD (e.g. the HSM) shall determine the current transaction key used by any transaction-originating SCD from:

a) the non-secret information contained in the transaction's Key Serial Number (KSN);

b) a Base Derivation Key.

The Base Derivation Key:

— is used in one or more receiving SCDs (e.g. at the Acquirer);

— does not exist in any transaction-originating SCD (e.g. at the point of sale);

— is used to generate the transaction-originating SCD's unique initial DUKPT key using a portion of the Key Serial Number;

— is used to generate the current transaction key at the receiving SCD;

— can be used to generate the unique initial DUKPT keys for many originating SCDs.

For DUKPT (AES and TDEA), the bit or byte order is assumed to be such that the leftmost bit, decimal digit, hexadecimal digit or byte is the most significant and the rightmost bit, decimal digit, hexadecimal digit or byte is assumed to be the least significant.

The Key Serial Number shall utilize a BDK Identifier (BDK ID). The Key Serial Number consists of three subfields. The leftmost subfield is a BDK Identifier, which is used to select the Base Derivation Key (BDK) appropriate for the SCD originating the transaction ("originating SCD").

The second subfield is a Derivation Identifier (DID). The BDK Identifier is concatenated with the Derivation Identifier and value is encrypted using the selected Base Derivation Key. The result is the initial DUKPT key that is loaded into the originating SCD at the time of initialization (though subsequently erased).

These first two subfields are collectively known as the Initial Key ID.

The third subfield is a transaction counter. The originating SCD shall increase its transaction counter for each transaction.

Once the transaction counter associated with an initial DUKPT key reaches its maximum, the originating SCD shall cease the use of any keys associated with that initial DUKPT key except for the purpose of computing the key encrypting key that encrypts a new initial DUKPT key. The receiving system should verify that the originating SCD's transaction counter has increased.

The initial DUKPT key and the transaction counter are inputs to a key derivation process that produces the transaction key used for the current transaction. In the specification of this method, the transformation process requires no more than 16 derivation cycles even though the transaction counter can have more than a billion different values.

Observe that the initially loaded key is a function of the Base Derivation Key, the BDK Identifier and the Derivation Identifier. Therefore, no two originating SCDs will be given the same initial DUKPT key provided that no two originating SCDs with the same BDK Identifier have identical Derivation Identifiers.

This method shall operate at the transaction-receiving SCD, as shown in Figure 4.

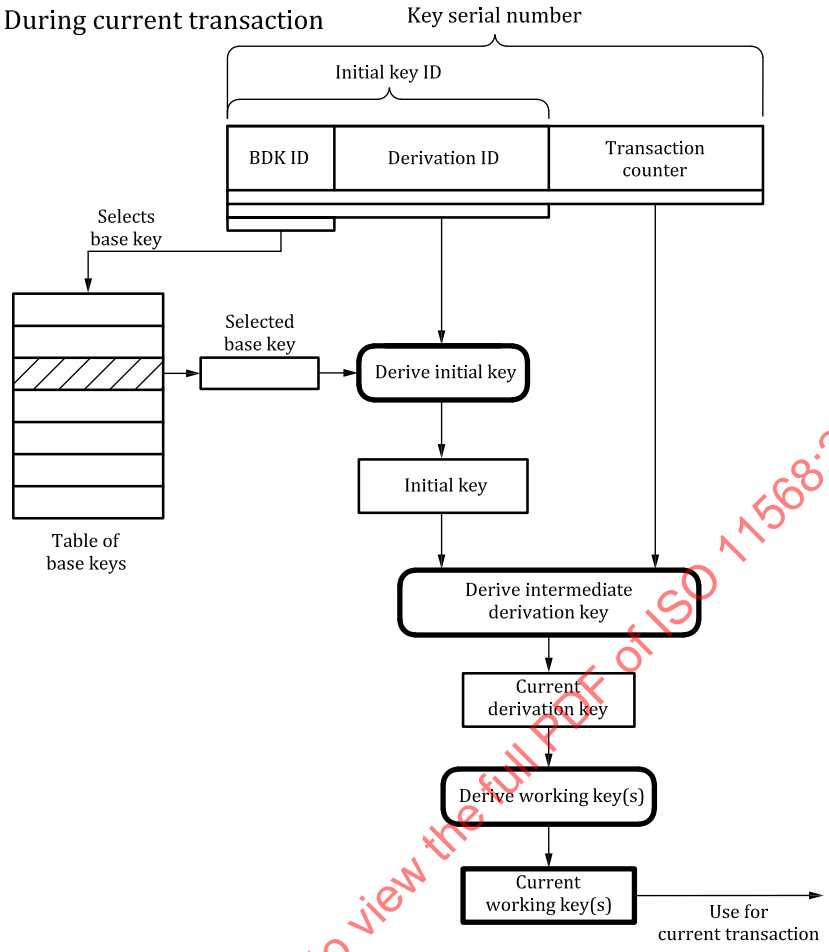


Figure 4 — DUKPT at transaction-receiving SCD

The originating SCD shall calculate and use a current transaction key using the process shown in Figure 5, such that the receiving SCD can determine the current transaction key using the process shown in Figure 4. The originating SCD shall also erase all record of the current transaction key immediately after completion of the current transaction.

An example of how this method operates at the originating SCD is shown in Figure 5. This SCD stores a number of intermediate derivation keys. At the beginning of a new transaction, the transaction counter (the rightmost portion of the Key Serial Number) is incremented and then is used to select one of these intermediate derivation keys as the current transaction key. The selected key is erased from future-key storage. Observe that the Key Serial Number is transmitted with the current transaction.

At the completion of the transaction, a number (sometimes none, sometimes one or more) of intermediate derivation keys are derived from the current transaction key as a function of the transaction counter. These newly generated intermediate derivation keys are then stored into those locations in future-key storage determined by the transaction counter. The current transaction key is then erased. Therefore, the SCD retains no information about any key used for any previous transaction.

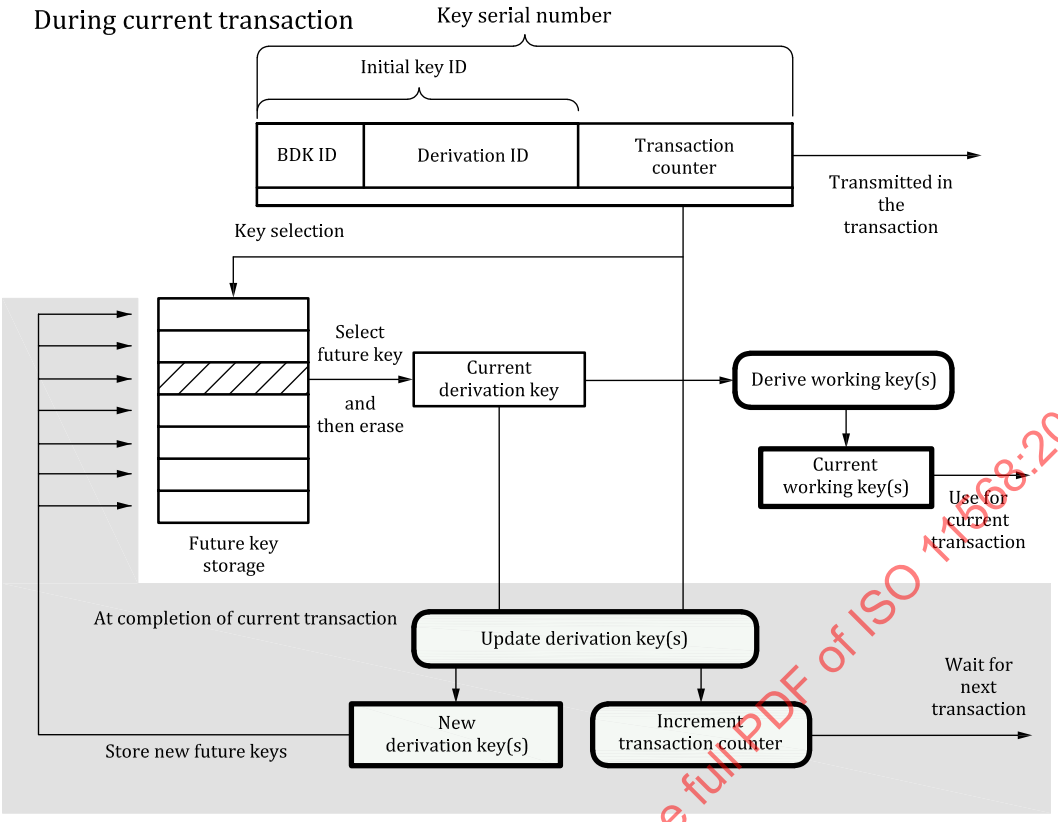


Figure 5 — DUKPT at transaction-originating SCD

The derivation processes 'A' and 'B' are different but related. Intermediate derivation keys are generated, stored and selected at the originating SCD in a manner such that the receiving SCD is able to determine the current transaction key.

Both the initial DUKPT key and the associated Key Serial Number shall be loaded into the originating SCD. The originating SCD shall ensure that the transaction counter is set to zero.

5.3.3 Unique initial keys

Originating SCDs that are initialized with an initial DUKPT key derived from the same Base Derivation Key (BDK) shall not be initialized using the same Derivation Identifier (DID). When the same Base Derivation Key is installed in more than one key loading SCD, and the Derivation Identifier is based on a counter, a mechanism shall exist to ensure that the originating SCDs initialized with an initial DUKPT key by each key loading SCD do not have the same initial DUKPT key (e.g. using an Derivation Identifier range for each instance of the key in one or more KLD or varying a portion of the BDK Identifier to identify each instance of the same key).

Initial DUKPT keys that have been injected into a device shall not be reused, even with the same device.

If the Derivation Identifier is a counter that increments at each injection, recovery from backup shall include a mechanism to ensure that Initial Key IDs that have been injected since the last backup are not reused.

The initial DUKPT key shall be of the same key size as the BDK from which it is derived.

5.3.4 AES DUKPT

5.3.4.1 General

AES DUKPT can generate AES and TDEA keys using an AES key derivation function and can be used to derive PIN encryption keys, MAC keys, data encryption keys, key encryption keys and key derivation keys. The technique as described supports all AES key sizes (128, 192, and 256 bits).

The AES version of DUKPT is similar to the original DEA or TDEA based DUKPT algorithms. Changes take advantage of advances in cryptographic techniques and improvements in computer processing speeds and memory sizes. The differences between AES and TDEA DUKPT are as follows:

a) The DEA-based one-way function used for key derivation is replaced with a key derivation function based on NIST SP 800-108 and using AES-ECB[1)] as the underlying function.

b) In the AES technique, all keys are derived using the same key derivation function. The TDEA technique supports four different key derivation techniques: triple-DEA for deriving the initial DUKPT key, a one-way function for register keys, variants for PIN and MAC keys and a combination of TDEA and variants for deriving data encryption keys.

c) The Key Serial Number (KSN) is 96 bits rather than 80 bits. The KSN is made up of the Initial Key ID plus the transaction counter.

d) The Initial Key ID (IKID) is 64 bits rather than 59 bits. The IKID is made up of a 32-bit BDK ID plus a 32-bit Derivation ID (DID). Formerly this was known as an initial Key Serial Number which was made up of a 40-bit key set index and 19-bit device identifier.

e) The transaction counter (TC) is 32 bits rather than 21 bits.

f) The algorithm supports transaction-originating devices with either 21 or 32 key registers.

g) Support for loading a new initial key under an existing key is included.

AES DUKPT will sometimes be installed in networks initially designed for DEA DUKPT. For those environments, format changes often need to be minimized, making it undesirable to increase the size of the Key Serial Number (KSN, the concatenation of the Initial Key ID and the transaction counter). The following description includes discussion of a KSN compatibility mode that can use the 80-bit KSN of DEA and TDEA DUKPT. Observe that the block size of AES is twice the size of DEA, so KSN compatibility mode does not guarantee compatibility with existing network formats.

This document describes the AES DUKPT algorithm and specifies functions for transaction-originating devices (e.g. in POS terminal) and transaction-receiving SCD (e.g. HSM). Either the methodology as described or its functional equivalent is performed to ensure that the Key Serial Number and keys are generated correctly.

5.3.4.2 AES DUKPT description

DUKPT provides a unique key per transaction where no information about any previous key is retained in the transaction-originating devices. To aid in understanding DUKPT, a simplified example is provided. This example describes a system that does not include a full DUKPT implementation. This simplified system operates as shown in I.4.

When the initial key is injected into the transaction-originating device, it is stored with the Initial Key ID (the concatenation of the BDK ID and Derivation ID) and a transaction counter, which is initialized to zero. This counter increments after each transaction. Each time the transaction counter increments, it is encrypted using the old key and the result becomes the new key. Thus, the new key is a non-reversible transformation of the old key. Given knowledge of the new key, there is no feasible way to determine the old key.

1) AES is used as pure block cipher at this point. Since only single blocks are encrypted, this is equivalent to AES-ECB encryption.

In every message, the transaction-originating device transmits the concatenation of the Initial Key ID and the transaction counter. The Acquirer is able to determine the initial key because it uses the base derivation key in its SCD and knows the device-specific information that is used in the derivation process. Having done this, the Acquirer is then able to perform the process indicated in Figure 5 with a counter value (binary) of ...000001, then repeat this process for a counter value of ...000010, then ...000011, etc., up to the value of the transaction counter included in the message. The result of these multiple passes through the process is the Current Key used to derive PIN, data encryption or MAC keys as needed.

This procedure, though theoretically possible, is clearly not feasible. The expected life of a transaction-originating device is a million or more PIN encryptions, so an excessive number of encryption cycles would be required by the transaction-receiving device to derive the Current Key once the transaction counter had reached a reasonably high count.

At the other end of the scale, the transaction-receiving device could derive each individual key by encrypting the counter with the initial key. But to meet the goal of not having any old key material retained in the transaction-originating devices, this would require it to pre-calculate and store every key that it will need in the future, which is infeasible for so many keys.

What is needed is a "key transformation" technique that the receiving device can follow without calculating all of the intervening keys. That is, the device is able to take "giant steps" forward and not regenerate every single intervening key to reach the current one, while the transaction-originating device can still step through a number of possibilities between each "giant step" so that the secure storage requirements are not unduly burdensome.

There are several ways in which this objective can be accomplished. The method chosen here is to make each key derivation a function not (necessarily) of the immediately preceding key but rather of that key for which the transaction counter contained the same bit configuration less the rightmost "one" bit. In other words:

The key corresponding to transaction counter value:	Is a non-reversible transformation of the key corresponding to transaction counter value:
0000 1011 0011	0000 1011 0010
0000 1011 0010	0000 1011 0000
0011 0000 1000	0011 0000 0000
etc.	etc.

As a result, a number of keys may all be the non-reversible transformation of the same key. For example, the keys corresponding to transaction counter values:

...0110 0101 0100 0001

...0110 0101 0100 0010

...0110 0101 0100 0100

...0110 0101 0100 1000

...0110 0101 0101 0000

...0110 0101 0110 0000

are all non-reversible transformations of the key corresponding to transaction counter value:

...0110 0101 0100 0000.

Each of the six listed transaction counter values has a unique key associated with it, even though all six resulting keys are based on the same key. To describe how this is accomplished, we will define:

K-A as the key associated with ...0110 0101 0100 0000.

and:

K-1 as the key associated with ...0110 0101 0100 0001;

K-2 as the key associated with ...0110 0101 0100 0010;

K-3 as the key associated with ...0110 0101 0100 0100;

K-4 as the key associated with ...0110 0101 0100 1000;

K-5 as the key associated with ...0110 0101 0101 0000;

K-6 as the key associated with ...0110 0101 0110 0000;

then:

K-1 = ...0110 0101 0100 0001 encrypted under K-A;

K-2 = ...0110 0101 0100 0010 encrypted under K-A;

K-3 = ...0110 0101 0100 0100 encrypted under K-A;

K-4 = ...0110 0101 0100 1000 encrypted under K-A;

K-5 = ...0110 0101 0101 0000 encrypted under K-A;

K-6 = ...0110 0101 0110 0000 encrypted under K-A.

In this way, K-1 to K-6 are each unique keys but each is a non-reversible transformation of the same key, K-A.

Observe that K-A will be used and thus is erased from the transaction-originating device before any of the keys K-1 to K-6 are used. K-1 to K-6 (in this example) are generated and stored for future use before K-A is used and erased. Therefore, the transaction-originating device is capable of storing a number of such intermediate derivation keys. The number of intermediate derivation keys that the transaction-originating device stores is equal to the number of binary bits in the transaction counter.

With this scheme, it is relatively easy for the Acquirer's security module to derive any key given its associated transaction counter value. After first deriving the initial key unique to the device, the module needs to perform only about as many encryption operations as there are "one" bits in the transaction counter value. For example, assume that the security module receives an encrypted PIN block along with a transaction counter value of 1010 1100 0010. (A shorter-than-actual counter is used to simplify the example.) Assuming that the security module determines that the initial key was "K-I", the security module then proceeds with the following steps:

a) encrypts 1000 0000 0000 using the key K-I;

b) encrypts 1010 0000 0000 using the result of Step 1;

c) encrypts 1010 1000 0000 using the result of Step 2;

d) encrypts 1010 1100 0000 using the result of Step 3;

e) encrypts 1010 1100 0010 using the result of Step 4;

f) derives the PIN key from the result of Step 5;

Thus, in this example, the security module has determined the PIN encryption key used to encrypt the PIN in question in only six encryption cycles.

The previous DUKPT standard only supported up to 1 million encryptions because a 21-bit transaction counter was used with a maximum of 10 one-bits set in the counter value at any time. In many

environments, a transaction-originating device could process more than a million transactions in its lifetime. AES DUKPT therefore offers three enhancements for extending the life of the device:

— AES DUKPT includes the option to derive a key encryption key called the DUKPT update key, so that the host can send a device a new initial key encrypted under that key. A new Initial Key ID is also sent.

— AES DUKPT includes the option to use transaction counters with up to 16 one-bits, instead of only transaction counters with 10 or fewer one-bits.

— AES DUKPT increases the size of the transaction counter of 32 bits with a maximum of 16 bits with value one.

Once a device starts producing transaction keys using one of these settings, it shall continue using the same settings until a new key is loaded.

5.3.5 KSN compatibility mode

This document defines a new format for the KSN, which is 16 bits longer than the legacy TDEA KSN. In cases where network infrastructure and existing software cannot easily accommodate the longer value, but designers would still like to take advantage of the increased cryptographic strength offered by the new version, this subclause defines an 80-bit KSN that is network-compatible. Observe that it is not cryptographically compatible; it uses AES for key derivation, so it will not derive the same key(s) even if the same KSN is used.

To recap, the new 96-bit KSN has the following structure:

Initial Key ID (64 bits)	Transaction counter (32 bits)

The legacy KSN had the following structure:

Legacy Initial key ID (59 bits)	Transaction counter (21 bits)

For KSN compatibility mode, when the KSN is processed by the originating and receiving SCDs, the KSN will have the following internal format:

Padding (4 bits)	Legacy Initial key ID (59 bits)	Padding (1 bit)	Padding (11 bits)	Transaction counter (21 bits)

All padding bits will be binary zero. The Initial Key ID is lined up on a 4-bit boundary so that the hexadecimal representations can easily be compared.

Observe that the padding transforms the 59-bit legacy Initial Key ID into a 64-bit Initial Key ID and the legacy 21 bit transaction counter into a 32-bit transaction counter.

It is up to the implementation whether maximum number of one-bits in the counter is 10 or 16.

Observe that the originating SCD will include the padding bits during the internal processing but will strip them for transmission. Likewise, the receiving SCD will add them back prior to deriving the initial DUKPT key.

There is no entirely unambiguous way for the acquiring host to distinguish an AES 80-bit KSN from a TDEA one. Consequently, the acquiring host will make this determination based on the algorithm of the BDK identified by the Initial Key ID and supply this information to the HSM.

Where necessary or useful, it is recommended that legacy Initial Key ID starting with the byte “0E” should be reserved for use with compatibility mode.

EXAMPLE

Key Set Identifier = 0E11111111

Device ID = 22222

Initial Key ID = 0E1111111122222

Legacy KSN = 0E111111112222200000

Internal KSN = 00E1111111122222000000000

5.3.6 Derived key OIDs

Somewhere in the transaction message, it may be useful to include information about the type and length of key that was derived to encrypt the transaction (e.g. AES-128, AES-256, 3-key TDEA). Use of the following reserved object identifiers (OIDs) is recommended:

1.3.133.16.840.9.24.1.1 – AES-128

1.3.133.16.840.9.24.1.2 – AES-192

1.3.133.16.840.9.24.1.3 – AES-256

1.3.133.16.840.9.24.1.4 – 2-key TDEA

1.3.133.16.840.9.24.1.5 – 3-key TDEA

Relative OIDs (ROIDS) can be used to save space in the transaction message. Use of these OIDs indicate the transaction key has the given algorithm and size and was derived according to the specification in this document.

5.3.7 Keys and key sizes

The derivation function is used to derive intermediate derivation keys from the initial DUKPT key, and those intermediate derivations keys are used to generate working keys for transactions. The initial DUKPT key shall be the same key size as the BDK from which it is derived. Intermediate derivation keys (and the current intermediate derivation key) shall be of the same size as the BDK and initial DUKPT key from which they are derived. Working keys shall be the same strength or weaker than the key from which they are derived. Legacy TDEA DUKPT did not allow for different working key sizes. Table 3 shows acceptable working key sizes based on the AES BDK under which they are derived.

Table 3 — Working key derivation matrix for AES BDK / IK

Derived Working Key	AES-128 BDK/IK	AES-192 BDK/IK	AES-256 BDK/IK
2TDEA working key	Allowed	Allowed	Allowed
3TDEA working key	Allowed	Allowed	Allowed
AES-128 working key	Allowed	Allowed	Allowed
AES-192 working key	Not allowed	Allowed	Allowed
AES-256 working key	Not allowed	Not allowed	Allowed
HMAC-128 working key	Allowed	Allowed	Allowed
HMAC-192 working key	Not allowed	Allowed	Allowed
HMAC-256 working key	Not allowed	Not allowed	Allowed

Working keys shall be interpreted to have a mode of use as defined in Table 4, for each working key with key usage indicator as indicated in bytes 2 to 3 of Table 5. Observe that unique per-transaction working keys are not normally stored as key blocks (because they are not stored outside the SCD) but some implementations may choose to do so. The mode of use is offered as a reference to specifying mode of use for DUKPT working keys and supporting mode of use enforcement by implementations regardless of whether key blocks are used.

In Table 4 the transaction-originating SCD is referred to as the client and the transaction-receiving SCD is referred to as the server or host.

Table 4 — Mode of use for working keys

AES DUKPT Working key usage	Key usage indicator value	Client TR-31 mode of use	Server or host TR-31 mode of use
Key encryption key	0x0002	'B' (0x42) Both wrap and unwrap	'B' (0x42) Both wrap and unwrap
PIN encryption	0x1000	'E' (0x45) Encrypt only	'D' (0x44) Decrypt only
Message authentication #1, client to server/host	0x2000	'G' (0x47) Generate only	'V' (0x56) Verify only
Message authentication #2, server to client	0x2001	'V' (0x56) Verify only	'G' (0x47) Generate only
Message authentication #3, bi-directional between client and server	0x2002	'C'(0x43) Both generate and verify	'C'(0x43) Both generate and verify
Data encryption #1, client to server	0x3000	'E' (0x45) Encrypt only	'D' (0x44) Decrypt only
Data encryption #2, server to client	0x3001	'D' (0x44) Decrypt only	'E' (0x45) Encrypt only
Data Encryption #3, bi-directional between client and server	0x3002	'B' (0x42) Both encrypt and decrypt	B' (0x42) Both encrypt and decrypt
Used in DUKPT derivation function	0x8000		
Used in DUKPT derivation function	0x8001		
Reserved for future ISO use	0000 -EFFF not assigned		
Reserved for proprietary use	F000 -FFFF		

5.3.8 Helper functions and definitions

5.3.8.1 Enumerations

The following enumerations define some data types used throughout the AES DUKPT function pseudocode.

```
// DerivationPurpose identifies if this derivation is to create an initial key
// or any other key type to help select which derivation data table to use
enum DerivationPurpose
{
   _Initial-Key_,
   _Derivation-or-Working-Key_
}

// KeyUsage defines the possible key usages that can be derived
enum KeyUsage
{
   _Key-Encryption-Key_,
   _PIN-Encryption_,
   _Message-Authentication-generation_,
   _Message-Authentication-verification_,
   _Message-Authentication-both-ways_,
   _Data-Encryption-encrypt_,
   _Data-Encryption-decrypt_,
   _Data-Encryption-both-ways_,
   _Key-Derivation_,
   _Key-Derivation-Initial-Key_
}

// KeyType defines the cryptographic key type being derived
enum KeyType
```

```
{
   _2TDEA_,
   _3TDEA_,
   _AES128_,
   _AES192_,
   _AES256_
}
```

5.3.8.2 Key length function

A helper function Key_Length is used to convert the key type into a key length (in bits).

```
Key_Length(keyType)
{
   if (keyType == _2TDEA_)       { return 128; }
   else if (keyType == _3TDEA_)  { return 192; }
   else if (keyType == _AES128_) { return 128; }
   else if (keyType == _AES192_) { return 192; }
   else if (keyType == _AES256_) { return 256; }
}
```

5.3.9 Key derivation function algorithm

Subclause 5.3.2 describes the DUKPT key derivation process as an encryption of the counter to help simplify understanding of the concept. In fact, AES DUKPT uses the Key Derivation Function (KDF) defined in NIST SP 800-108 in Counter Mode with AES-ECB as the underlying function for deriving keys. This function is illustrated in Figure 6.

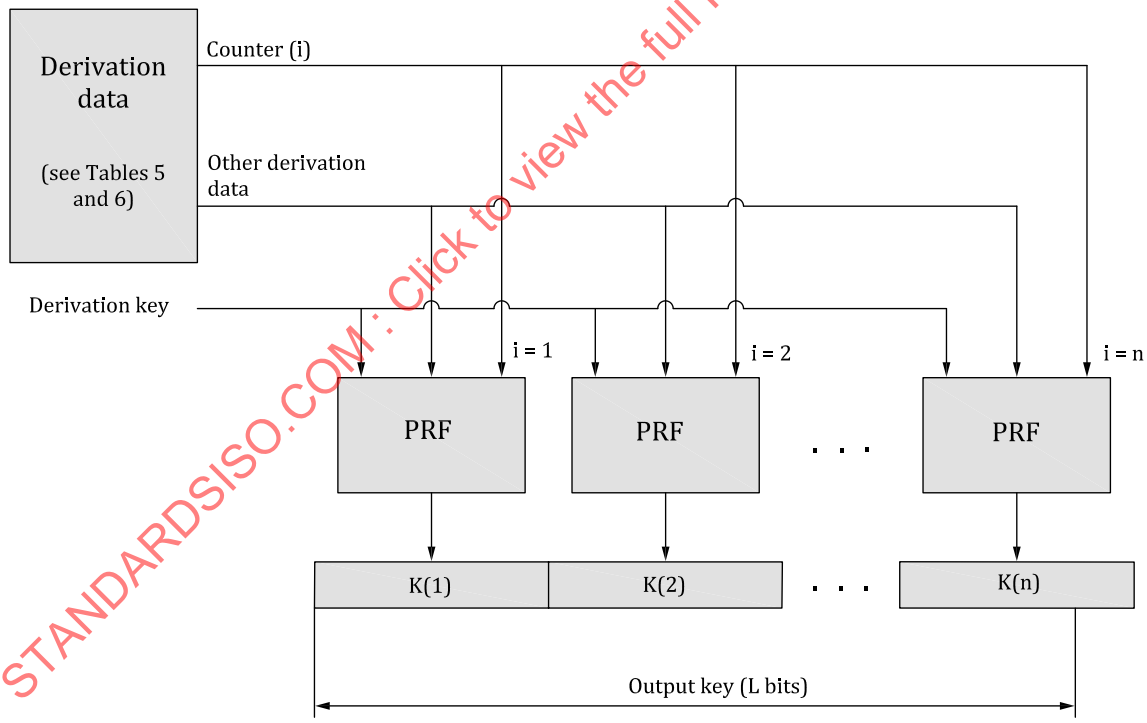


Figure 6 — KDF in Counter Mode

NOTE 1 See 5.3.10 for details on derivation data.

NOTE 2 PRF is a "pseudo random function". It produces a random appearing value from the input values.

NOTE 3 Output key length can be such that only part of the last block is used.

The following values are inputs to this function.

Derivation data: The non-secret data used to derive a key. This is the data that is encrypted. This data is defined in 5.3.10.

Derivation key: The secret value used as a key in the derivation function. For this application, the derivation key is the AES key that encrypts the derivation data. A derivation key may be the result of a previous call to the derivation function.

Derivation key length: The length of the derivation key.

L, the output key length: The length in bits of the output key (the key being created). This value will be 128, 192, or 256. The output is an AES or TDEA key.

"Derive key" (local subroutine): The Derive_Key function takes the derivation key, the desired output length of the key to be derived and the derivation data (defined in the next subclause) and outputs a derived key equal in length to the derivation key.

```
Derive_Key(derivationKey, keyType, derivationData)
{
   L = Key_Length(keyType);
   n = ceil(L/128); // number of blocks required to construct the derived key
   for (i=1;i<=n;i++) {
      // Set the value of the derivation data key block counter field equal to
      // the block count being derived.
      // First block is 0x01, second block is 0x02.
      derivationData[1] = i;
     result[(i-1)*16..i*16-1] = AES_Encrypt_ECB(derivationKey, derivationData);

   }
   derivedKey = result[0..(L/8)-1];
   return derivedKey;
}
```

5.3.10 Derivation data

Derivation data is used with the Derive_Key function to derive keys of a specific type. Tables 5 and 6 specify 16 bytes of derivation data which fits into a single block of the AES cryptographic primitive used in the key derivation algorithm previously defined.

Table 5 shows the derivation data for creating the initial key for a transaction-originating device. Table 6 is used for the generation of all other keys. The main difference between the two tables is in the last 8 bytes. For the derivation data used in creating the initial key Table 5 uses the full Initial Key ID for the last 8 bytes which is needed to provide a unique initial key. For deriving all other keys Table 6 uses the rightmost 4 bytes of the Initial Key ID plus the 4 bytes of the transaction counter.

Table 5 — Originating SCD key derivation data

Byte #	Field name	Description	Encoding	Range of values
0	Version	Version ID of this table structure. This edition is represented as version 1 represented by 0x01.	2H	0x01
1	Key Block Counter	A counter that is incremented for each 16-byte block of keying material generated for a pair of encryption and MAC keys. Starts at 1 for each key being generated.	2H	0x01 – 0x02
2 to 3	Key Usage Indicator	Indicates how the key to be derived is to be used. The initial key is always a key derivation key.	4H	0x8001 = Key Derivation, Initial key
4 to 5	Algorithm Indicator	Indicates the encryption algorithm that is going to use the derived key.	4H	0x0002 = AES 128 bit 0x0003 = AES 192 bit 0x0004 = AES 256 bit

Table 5 *(continued)*

Byte #	Field name	Description	Encoding	Range of values
6 to 7	Length	Length, in bits, of the keying material being generated.	4H	0x0080 if 128 bits is being generated (AES-128) 0x00C0 if 192 bits is being generated (AES-192) 0x0100 if 256 bits is being generated (AES-256)
8 to 15	Initial Key ID	The originating SCD's Initial Key ID, the leftmost 64 bits of the Key Serial Number.	16H	0x0000000000000000 to 0xFFFFFFFFFFFFFFFF

Table 6 — Other key derivation data

Byte #	Field Name	Description	Encoding	Range of Values
0	Version	Version ID of this table structure. This edition is represented as version 1 represented by 0x01	2H	0x01
1	Key Block Counter	A counter that is incremented for each 16-byte block of keying material generated for a pair of encryption and MAC keys. Starts at 1 for each key being generated.	2H	0x01 – 0x02
2 to 3	Key Usage Indicator	Indicates how the key to be derived is to be used	4H	Defined in Table 4
4 to 5	Algorithm Indicator	Indicates the algorithm that is going to use the derived key.	4H	0x0000 = 2-key TDEA 0x0001 = 3-key TDEA 0x0002 = AES 128 bit 0x0003 = AES 192 bit 0x0004 = AES 256 bit 0x0005 = HMAC
6 to 7	Length	Length, in bits, of the keying material being generated.	4H	0x0080 if 128 bits is being generated (AES-128, 2TDEA, or 128-bit HMAC key) 0x00C0 if 192 bits is being generated (AES-192, 3TDEA, or 192-bit HMAC key) 0x0100 if 256 bits is being generated (AES-256 or 256-bit HMAC key)
8 to 11	Initial Key ID	The rightmost half of the originating SCD's Initial Key ID.	8H	0x00000000 to 0xFFFFFFFF
12 to 15	Transaction Counter	The 32-bit transaction counter	8H	0x00000000 to 0xFFFFFFFF

5.3.11 "Create Derivation Data" (local subroutine)

The Create_Derivation_Data function creates the derivation data according to Table 5 or Table 6 depending on the "derivationPurpose". The inputs of keyUsage and derivedKeyType help define derivation data for the key being derived, and initialKeyID and transaction counter are also used in the derivation data block.

```
Create_Derivation_Data(derivationPurpose, keyUsage, derivedKeyType, initialKeyID,
                       counter)
{
   // Set Version ID of the table structure.
```

```
   DerivationData[0] = 0x01; // version 1

   // set Key Block Counter
   DerivationData[1] = 0x01; // 1 for first block, 2 for second, etc.

   // set Key Usage Indicator
   if (keyUsage == _Key-Encryption-Key_) {
      DerivationData[2..3] = 0x0002; }
   else if (keyUsage == _PIN-Encryption_) {
      DerivationData[2..3] = 0x1000; }
   else if (keyUsage == _Message-Authentication-generation_) {
      DerivationData[2..3] = 0x2000; }
   else if (keyUsage == _Message-Authentication-verification_) {
      DerivationData[2..3] = 0x2001; }
   else if (keyUsage == _Message-Authentication-both-ways_) {
      DerivationData[2..3] = 0x2002; }
   else if (keyUsage == _Data-Encryption-encrypt_) {
      DerivationData[2..3] = 0x3000; }
   else if (keyUsage == _Data-Encryption-decrypt_) {
      DerivationData[2..3] = 0x3001; }
   else if (keyUsage == _Data-Encryption-both-ways_) {
      DerivationData[2..3] = 0x3002; }
   else if (keyUsage == _Key-Derivation_) {
      DerivationData[2..3] = 0x8000; }
   else if (keyUsage == _Key-Derivation-Initial-Key_) {
      DerivationData[2..3] = 0x8001; }

   // set Algorithm Indicator and key size
   if (derivedKeyType == _2TDEA_) {       // Note: 2TDEA and 3TDEA are included
      DerivationData[4..5] = 0x0000; }    //       as an option for working keys
   else if (derivedKeyType == _3TDEA_) {  //       and shall not be used for
      DerivationData[4..5] = 0x0001; }    //       key derivation keys
   else if (derivedKeyType == _AES128_) {
      DerivationData[4..5] = 0x0002; }
   else if (derivedKeyType == _AES192_) {
      DerivationData[4..5] = 0x0003; }
   else if (derivedKeyType == _AES256_) {
      DerivationData[4..5] = 0x0004; }

   // set length of key material being generated
   if (derivationPurpose == _Initial-Key_) {
      DerivationData[6..7] = 0x0000; }
   else if (derivedKeyType == _2TDEA_) {
      DerivationData[6..7] = 0x0080; }
   else if (derivedKeyType == _3TDEA_) {
      DerivationData[6..7] = 0x00C0; }
   else if (derivedKeyType == _AES128_) {
      DerivationData[6..7] = 0x0080; }
   else if (derivedKeyType == _AES192_) {
      DerivationData[6..7] = 0x00C0; }
   else if (derived_key_type == _AES256_) {
      DerivationData[6..7] = 0x0100; }

   //  next 8 bytes depend on the derivation purpose
   if (derivationPurpose == _Initial-Key_) {
      DerivationData[8..15] = initialKeyID[0..7];
   }
   else if (derivationPurpose == _Derivation-or-Working-Key_) {
      DerivationData[8..11] = initialKeyID[4..7];
      DerivationData[12..15] = counter[0..3];
   }

   Return DerivationData;
}
```

5.3.12 Security considerations

The key derivation function defined in NIST SP 800-108 requires the use of a pseudo-random function and approves the use of the CMAC mode of operation for block ciphers to provide this functionality.

This document, however, uses AES ECB mode instead of using AES CMAC, despite the fact that AES ECB is not approved by NIST for this usage.

The rationale for this choice is performance. For most applications, keys are derived relatively infrequently, with the derived key being used for a number of operations once it is created. In DUKPT, the server derives the working key using an iterative process, with up to 18 calls to the key derivation function to process a single PIN block. A single host may be communicating with tens of thousands of input devices, each sending messages of only one or two blocks. Thus, the processing cost of the key derivation function is critical to the system performance. For an AES-128 key, using CMAC rather than ECB would cut the performance of the key derivation function roughly in half.

Despite the performance benefits, AES ECB is not approved in the NIST key derivation function because the NIST function needs to handle a variety of key derivation applications, with differing lengths and types of key derivation data. In DUKPT, we limit the derivation data to a single block. This single block limitation allows us to use ECB securely. The following are some frequently asked questions about the use of ECB instead of CMAC in this function.

— Isn't ECB a pseudo-random permutation rather than a pseudo-random function?

  Both ECB and CMAC are pseudo-random permutations when used with a single block of derivation data, and NIST puts no restrictions on CMAC-based derivations. Further, in order to distinguish between a pseudo-random function and a pseudo-random permutation, a key would need to be reused approximately 2^{64} times, which could only happen with a very detectable attack on the current network.

— Isn't CMAC more secure than ECB?

  As shown in the ISO/IEC 9798 series, ECB is a secure MAC on data when the length of that data is fixed to a single block length. CMAC's whitening step on the final block of data was added solely to improve security of the arbitrary length data use case. For DUKPT, arbitrary data lengths are not permitted.

— Doesn't CMAC prevent known plaintext attacks against the key derivation function?

  Since the output of the key derivation function is a key, and that key is never made public, known plaintext attacks are very difficult with either approach. CMAC could possibly make such attacks harder if the working keys are inappropriately revealed. However, there are no significant known plaintext attacks against AES. If such an attack is found, its impact on PIN block encryption would be profound. Protecting DUKPT against this theoretical attack wouldn't significantly improve the security of the overall network.

— Does CMAC make exhaustive search more difficult?

  Again, first the attacker would need to compromise (or recover by exhaustive search) the working key. From the working key, exhaustive search for the derivation key is very similar with both CMAC and ECB. For ECB, the attacker simply picks a key and tries encrypting the derivation data as a single ECB operation. For CMAC, the attacker picks a key and calculates the CMAC on that derivation data. The CMAC takes two calls to the AES block operation, so it would take about two times the effort to recover the key.

— DUKPT derives a large number of keys from very closely related derivation data, and those theoretically related keys could be used to encrypt the same data or closely related data. Is there a related key attack here that CMAC could mitigate?

  There are no known related key attacks where the keys are related in this manner. Basically, this implies that the encryption of two related plaintexts would need to result in some relationship in the ciphertext, which would seem to imply other types of attacks on AES. If this type of attack does exist, CMAC would not prevent the attack entirely, since, for example, the PIN key, data keys for multiple directions and MAC key would still be related keys even if the CMAC whitening obscured some of the related keys in the derivation tree.

5.3.13 Host security module algorithm

5.3.14 General

AES DUKPT is relatively simple to implement at the host side. Observe that the host does not need to know which transaction counter values the terminal is skipping in order to process a transaction; it simply calculates the appropriate keys for the value of transaction counter that was sent. Also, the only time the host needs to know the number of registers in a terminal is when sending a new initial key. The key encryption key is always derived from the highest transaction counter value supported by the terminal, which is a transaction counter value that the terminal has not used yet.

For KSN compatibility, the host simply separates the 80-bit KSN into 56 bits of Initial Key ID and 21 bits of transaction counter. The Initial Key ID is padded to the right with zeros to create 64 bits. The transaction counter is padded to the left with zeros to create 32 bits.

5.3.15 "Derive Initial Key"

The Derive_Initial_Key function sets the initial key derivation data up according to Table 5 using the Initial Key ID and the appropriate constants for the AES size being used.

EXAMPLE 1 For AES-128, Initial Key ID = 0123456789ABCDEF, the derivation data for the Initial Key is constructed using Table 5 as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Initial Key ID
01	01	8001	0002	0080	0123456789ABCDEF

Where the resulting Initial Key derivation data = 0101800100020080001234567 89ABCDEF.

This derivation data would be encrypted with the AES-128 BDK where the result would be the AES 128-bit Initial Key for this Initial Key ID.

EXAMPLE 2 For AES-256, Initial Key ID = FFEEDDCCBBAA9988, the derivation data for the Initial Key is constructed using Table 5 as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Initial Key ID
01	01	8001	0004	0100	FFEEDDCCBBAA9988
01	02	8001	0004	0100	FFEEDDCCBBAA9988

Where the resulting derivation data is:

1st block derivation data = 0101800100040100FFEEDDCCBBAA9988.

2nd block derivation data = 0102800100040100FFEEDDCCBBAA9988.

The derivation data for each block would be encrypted with the AES-256 BDK where the result would be two 128-bit data blocks that are concatenated to create the AES 256-bit Initial Key for this Initial Key ID.

```
Derive_Initial_Key(BDK, keyType, initialKeyID)
{
   derivationData = Create_Derivation_Data(_Initial-Key_,
                                           _Key-Derivation-Initial-Key_,
                                           keyType, initialKeyID, 0);

   initialKey = Derive_Key(BDK, keyType, derivationData);

   return initialKey;

}
```

5.3.16 "Host Derive Working Key"

The Host_Derive_Working_Key function starts with the Base Derivation Key (*BDK*) and calculates from that the initial key using the *initialKeyID* and *deriveKeyType* (one of _AES128_, _AES192_, _AES256_, _2DEA_, or _3DEA_). From the initial key the function calculates current derivation key specified by the *transactionCounter* by deriving all the intermediate derivation keys in the derivation tree from the initial key based on the current *transactionCounter* value. Finally the *workingKey* is calculated as derived from the current derivation key.

```
Host_Derive_Working_Key(BDK, deriveKeyType, workingKeyUsage, workingKeyType, initialKeyID,
                        transactionCounter)
{
   initialKey = Derive_Initial_Key(BDK, deriveKeyType, initialKeyID);

   // set the most significant bit to one and all other bits to zero
   mask = 0x80000000;
   workingCounter = 0;
   derivationKey = initialKey;

   // calculate current derivation key from initial key
   while (mask > 0) {
      if ((mask AND transactionCounter) != 0) {
         workingCounter = workingCounter OR mask;
         derivationData = Create_Derivation_Data(_Derivation-or-Working-Key_,
                                                 _Key-Derivation_,
                                                 deriveKeyType, initialKeyID,
                                                 workingCounter);
         derivationKey = Derive_Key(derivationKey, deriveKeyType, derivationData);
      }
      mask = mask >> 1;
   }

   // derive working key from current derivation key
   derivationData = Create_Derivation_Data(_Derivation-or-Working-Key_, workingKeyUsage,
                                           workingKeyType, initialKeyID,
                                           transactionCounter);
   workingKey = Derive_Key(derivationKey, workingKeyType, derivationData);

   return workingKey;

}
```

5.3.17 Intermediate derivation key derivation data examples

For an example of derivation data for deriving intermediate derivation keys, set the initial derivation data according to Table 6, and construct the derivation data as follows:

EXAMPLE 1 For AES 128 where the Initial Key ID = 0123456789ABCDEF, the derivation data for the intermediate derivation key where the counter value is 0x00000001 is constructed as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Right half of Initial Key ID	Transaction Counter
01	01	8000	0002	0080	89ABCDEF	00000001

Where the resulting derivation data = 01018000000200 8089ABCDEF00000001.

This derivation data would be encrypted with the Initial Key where the result would be the AES 128-bit intermediate derivation key for counter value 0x00000001.

EXAMPLE 2 For AES 256 where the Initial Key ID = FFEEDDCCBBAA9988, the derivation data for the two 128-bit blocks used to create the AES 256-bit intermediate derivation key where the counter value is 0x00000001 is constructed as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Right half of Initial Key ID	Transaction Counter
01	01	8000	0004	0100	BBAA9988	00000001
01	02	8000	0004	0100	BBAA9988	00000001

Where the resulting derivation data is

1st block derivation data = 0101800000040100BBAA998800000001.

2nd block derivation data = 0102800000040100BBAA998800000001.

The derivation data for each block would be encrypted with the Initial Key where the result would be two 128-bit data blocks that are concatenated to create the AES 256-bit intermediate derivation key for counter value 0x0000001.

EXAMPLE 3 This derivation data is updated with each derivation data calculation. In the AES-128 example, if the working counter is 0x00001200, then the derivation data is constructed as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Right half of Initial Key ID	Transaction Counter
01	01	8000	0002	0080	89ABCDEF	00001200

Where the resulting derivation data = 0101800000020089ABCDEF00001200.

This derivation data would be encrypted with the intermediate derivation key for counter value 0x00001000, where the result would be the AES 128-bit intermediate derivation key for counter value 0x00001200. The reason the intermediate derivation key for counter value 0x00001000 is used to derive this key is that this is the parent key for intermediate derivation key for counter value 0x00001200. This is calculated by masking the rightmost 1 bit of 0x00001200 and the result is 0x00001000.

5.3.18 Working key derivation data examples

For an example of derivation for deriving working keys from the current derivation key, set the derivation data using the values for the appropriate key usages from Table 6 and construct the derivation data as follows:

EXAMPLE 1 Data Encryption, encrypt key, AES-128, Initial Key ID = 0123456789ABCDEF, transaction counter = 0x00002345. The derivation data is constructed as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Right half of Initial Key ID	Transaction Counter
01	01	3000	0002	0080	89ABCDEF	00002345

Where the resulting derivation data = 0101300000020089ABCDEF00002345.

This derivation data would be encrypted with the intermediate derivation key for counter value 0x00002345 where the result would be the AES 128-bit data encryption, encrypt key for counter value 0x00002345.

EXAMPLE 2 Same device, PIN Encryption key, 2-key TDEA, Initial Key ID = 0123456789ABCDEF, transaction counter = 0x00002345. The derivation data is constructed as follows:

Version	Key Block Counter	Key Usage Indicator	Algorithm Indicator	Length	Right half of Initial Key ID	Transaction Counter
01	01	1000	0000	0080	89ABCDEF	00002345

Where the resulting derivation data = 0101100000000089ABCDEF00002345.

This derivation data would be encrypted with the intermediate derivation key for counter value 0x00002345 where the result would be the 2-key TDEA PIN Encryption key for counter value 0x00002345.

5.3.19 Transaction-originating device algorithm

5.3.19.1 Algorithm parameters

5.3.19.1.1 KEYLENGTH

This value is the length of the initial key. All keys stored in intermediate derivation key registers will be the same length as the initial key. PIN, MAC or data encryption keys can have different key lengths.

5.3.19.1.2 NUM_REG

This value is the number of intermediate derivation key registers. In TDEA DUKPT, this value was always 21. AES DUKPT uses 32 registers. AES DUKPT uses a value of 21 when using backward compatibility mode.

5.3.19.1.3 MAX_WORK

This is the maximum number of operations that it takes the host security module to arrive at the key for the current transaction counter value. This factor is determined by the number of ones in the transaction counter. In TDEA DUKPT, this value is always 10. For AES DUKPT the MAX_WORK value is 16. AES DUKPT uses a value of 10 when using backward compatibility mode.

5.3.19.2 Storage areas

5.3.19.2.1 General

The transaction-originating device maintains the following storage areas from the time of the "Load Initial Key" command for the life of the device.

5.3.19.2.2 Initial Key Identifier (64 bits)

This is defined in the pseudocode notation as the global variable *InitialKeyID*. In systems without KSN compatibility requirements, this value is injected as a 64-bit Initial Key ID. In devices with KSN compatibility, this register will hold the leftmost 64 bits of the initial Key Serial Number injected into the transaction-originating device along with the initial key during the "Load Initial Key" command. Observe that these 64 bits include bits from the transaction counter, but since the counter is zero, this approach effectively pads the 59-bit Initial Key ID with zeros on the right. The contents of this register remain fixed for the service life of the device, until another "Load Initial Key" command or until an “Update Initial Key” command is received.

5.3.19.2.3 Transaction counter (32 bits)

This is defined in the pseudocode notation as the global variable *TransactionCounter*. This variable defines a counter of the number of transactions that have occurred since the initial key was first loaded. Certain counter values are skipped (as explained) thus this value will not always represent the actual number of transactions that have been performed. Observe that the maximum value for the transaction counter is determined by the number of intermediate derivation key registers.

5.3.19.2.4 NUM_REG (32 or 21)

This is the number of intermediate derivation key registers. It is 32, except in compatibility mode where it is 21.

5.3.19.2.5 Intermediate derivation key registers (NUM_REG registers of KEYLENGTH)

This is defined in the pseudocode notation as a global array *IntermediateDerivationKeyRegister*. *IntermediateDerivationKeyRegister* is a set of registers used to store intermediate derivation keys. AES DUKPT defines 32 registers. When operating in backward compatibility mode, there will be 21 registers.

5.3.19.2.6 Intermediate derivation key in use (NUM_REG Booleans)

This is defined in the pseudocode notation as an array *IntermediateDerivationKeyInUse*. *IntermediateDerivationKeyInUse* is a set of Booleans used to store whether the corresponding *IntermediateDerivationKeyRegister* has a valid value.

The following storage areas relating to key management are required on a temporary basis and may be used for other purposes by other routines:

5.3.19.2.7 Current derivation key (0..31)

This is defined in the pseudocode notation as the global variable *CurrentDerivationKey*. This contains the index of the Intermediate Derivation Key Register whose contents are being used in the current cryptographic operation.

5.3.19.2.8 Shift register (32 bits)

This is defined in the pseudocode notation as the global variable *ShiftRegister*. This is a 32-bit register. This register normally contains 31 "zero" bits and a single "one" bit. This register is used to select one of the intermediate derivation key registers. Its value is 1 << *CurrentDerivationKey*.

5.3.19.2.9 Key type injected (KeyType)

This is the key type of the injected initial key, either _AES128_, _AES192_, or _AES256_.

5.3.19.3 Processing routines

5.3.19.3.1 General

Unlike the original annex that defined TDEA DUKPT, this document limits the description of the algorithm to the key management functions. PIN processing and entry are considered separate systems that may make use of the DUKPT key management. From this point of view, a device that implements AES DUKPT supports three key management functions that may be used by other device functions. These functions are Load_Initial_Key, Generate_Working_Keys and Update_Initial_Key. The processing steps to be performed in each case are indicated. This subclause also contains definitions of subroutines that are common to these functions. Observe that the key derivation function Derive_Key is defined in 5.3.9.

5.3.19.3.2 "Load Initial Key" (key management command)

The Load_Initial_Key function initializes the system from an initial key, the *initialKeyID*, and a *initialKeyType*. The *initialKeytype* shall be one of _AES128_, _AES192_, or _AES256_ and defines the type of the derivation keys to use (aka “intermediate derivation keys”).

```
Load_Initial_Key(initialKey, initialKeyType, initialKeyID)
{
   IntermediateDerivationKeyRegister[0] = initialKey;
   IntermediateDerivationKeyInUse[0] = true;
   CurrentDerivationKey = 0;
   InitialKeyID = initialKeyID;
   TransactionCounter = 0;
   ShiftRegister = 1;
   InitialKeyType = initialKeyType;
```

```
   Update_Derivation_Keys(NUM_REG-1);

   TransactionCounter++;
}
```

5.3.19.3.3 "Calculate DUKPT Update Key" (optional local subroutine)

The wrapping key for the Update_Initial_Key function is the key that corresponds to the last counter value. That counter value shall not be used to calculate transaction keys.

```
Calculate_DUKPT_Update_Key()
{
   bit = 1 << (NUM_REG - 1);
   register = NUM_REG - 1;

   while ((TransactionCounter & bit) == 1)
   {
      bit = bit >> 1;
      register--;
   }

   ctr = TransactionCounter & ~(bit - 1);
   key = IntermediateDerivationKeyRegister[register];

   while (bit > 0)
   {
      ctr = ctr | bit;
      derivationData = Create_Derivation_data(_Derivation-or-Working-Key_, _Key-
Derivation_, InitialKeyType, InitialKeyID, ctr);
      key = Derive_Key(key, deriveKeyType, derivationData);
      register--;
      bit = bit >> 1;
   }

   derivationData = Create_Derivation_Data(_Derivation-or-Working-Key_,
                                           _Key-Encryption-Key_,
                                           InitialKeyType, InitialKeyID,
                                           ctr);
   return Derive_Key(key, InitialKeyType, derivationData);
}
```

5.3.19.3.4 "Update Initial Key" (optional key management command)

The Update_Initial_Key function takes a new encrypted initial key wrapped with the last possible key encryption key derived from the current initial key on the device, decrypts the new initial key with the key encryption key and installs the new encryption key.

The new initial key shall be protected in a way that complies with the requirements in this document.

UnwrapKey shall be implemented in a way that conforms to the requirements in this document [e.g. by using a key block described in ISO 20038 (for AES) or ANSI X9.143 (for TDEA)].

The wrapping key shall only be used with a single key wrap algorithm and mode.

If TR-31 key blocks are used, the TR-31 key block carrying the initial key should have the following format:

```
D nnnn B1 A X 00 N 01 00 KS 14 hhhhhhhhhhhhhhhh
```

where nnnn is the decimal key block length (including count and header) and hhhhhhhhhhhhhhhh is the key identifier.

```
Update_Initial_Key(encryptedInitialKey, newInitialKeyType, newInitialKeyID)
{

   // derive a key encryption key from the last counter value
   // for the device
```

```
      dukptUpdateKey = Calculate_DUKPT_Update_Key();
   if (!UnwrapKey(encryptedInitialKey, dukptUpdateKey, newInitialKey))
   {
      return ERROR;
   }

   Load_Initial_Key(newInitialKey, newInitialKeyType, newInitialKeyID);

   return SUCCESS;
}
```

5.3.19.3.5 "Generate Working Keys" (key management command)

The Generate_Working_Keys function generates a working key for the current transaction.

Working keys shall only be used with a single algorithm and mode and for a single purpose.

```
Generate_Working_Keys(workingKeyUsage, workingKeyType)
{
   // initialize the ShiftRegister and CurrentDerivationKey
   Set_Shift_Register();

   // advance transaction counter until Current Key is valid
   while ( !IntermediateDerivationKeyInUse[CurrentDerivationKey] ){
      TransactionCounter = TransactionCounter + ShiftRegister; // skip over invalid key
      if (TransactionCounter >= ((1 << NUM_REG) - 1)) {
         Cease_Operation(); // this function will cease using this this key set
         return ERROR;
      }
      Set_Shift_Register();

   }

   // derive a working key from the Current Key pointer
   derivationData = Create_Derivation_Data(_Derivation-or-Working-Key_, workingKeyUsage,
                                           workingKeyType, InitialKeyID,
TransactionCounter);
   workingKey = Derive_Key(IntermediateDerivationKeyRegister[CurrentDerivationKey],
                           workingKeyType, derivationData);

   Update_State_for_next_Transaction();

   return workingKey;
}
```

5.3.19.3.6 "Update State for next Transaction" (local subroutine)

The Update_State_for_next_Transaction function invalidates the Current Key in the register named *IntermediateDerivationKeyRegister* and its in-use flag and increments the *TransactionCounter*. *InitialKeyType* is the key type for the injected initial key.

```
Count_One_Bits(value)
{
        // For clarity, a simple, straightforward implementation has been chosen.
    // There are faster ways of computing the value on most platforms.
    bits = 0;
    mask = 1 << (NUMREG - 1);
    while (mask > 0) {
        if ((value & mask) > 0)
            bits = bits + 1;
        mask = mask >> 1;
    }
    return bits;
}

Update_State_for_next_Transaction()
{
   oneBits = Count_One_Bits(TransactionCounter);
   if (oneBits <= MAX_WORK) {
      Update_Derivation_Keys(CurrentDerivationKey);
      // erase the current intermediate derivation key
```

```
      IntermediateDerivationKeyRegister[CurrentDerivationKey] = 0;
      // invalidate the current register
      IntermediateDerivationKeyInUse[CurrentDerivationKey] = false;
      // increment the transaction counter
      TransactionCounter++;
   }
   else { // number of "one" bits in the transaction counter is greater than MAX_WORK
      // erase the current intermediate derivation key
      IntermediateDerivationKeyRegister[CurrentDerivationKey] = 0;
      // invalidate the current register
      IntermediateDerivationKeyInUse[CurrentDerivationKey] = false;
      // skip transaction counter values with more than MAX_WORK bits
      TransactionCounter = TransactionCounter + ShiftRegister;
   }

   // check if transaction counter has exceeded max value
   if (TransactionCounter >= ((1 << NUM_REG) - 1)) {
      Cease_Operation(); // this function will cease using this this key set
      return ERROR;
   }

   return SUCCESS;
}
```

5.3.19.3.7 "Update Derivation Keys" (local subroutine)

The Update_Derivation_Keys function updates the intermediate derivation key registers based on the current values of the *ShiftRegister*, *TranactionCounter*, *CurrentDerivationKey*, *InitialKeyType* and *InitialKeyID*.

The first time this function is called, start = NUM_REG-1 and all the registers are initialized.

When it is called after a transaction key is generated, start = CurrentDerivationKey.

```
Update_Derivation_Keys(start)
{
   index = start;
   registerBit = 1 << start;
       baseKey = IntermediateDerivationKeyRegister[CurrentDerivationKey];
   while (registerBit != 0) {
      derivationData = Create_Derivation_Data(_Derivation-or-Working-Key_,
                                              _Key-Derivation_,
                                              InitialKeyType, InitialKeyID,
                                              (registerBit OR TransactionCounter));

      IntermediateDerivationKeyRegister[index] =  Derive_Key(
                           baseKey,
                           deriveKeyType, derivationData);
      IntermediateDerivationKeyInUse[index] = true;
      registerBit = registerBit >> 1;
      index = index - 1;
   }
   return SUCCESS;
}
```

5.3.19.3.8 "Set Shift Register" (local subroutine)

The Set_Shift_Register function sets to "one" that bit in the *ShiftRegister* that corresponds to the rightmost "one" bit in the *TransactionCounter*, making all other bits in the *ShiftRegister* equal zero. For example:

If Transaction Counter = 0000 0000 0000 0010 1100 1101 1100 0011,

Shift Register becomes: 0000 0000 0000 0000 0000 0000 0000 0001;

If Transaction Counter = 0000 0000 0000 0010 1100 1101 1100 0000,

Shift Register becomes: 0000 0000 0000 0000 0000 0000 0100 0000.

```
Set_Shift_Register()
{
   ShiftRegister = 1;
   CurrentDerivationKey = 0;

   if (TransactionCounter == 0) {
      Return SUCCESS;
   }

   While ((ShiftRegister AND TransactionCounter) == 0) {
      ShiftRegister = ShiftRegister << 1;
      CurrentDerivationKey++;
   }

   Return SUCCESS;
}
```

5.3.19.4 Base cipher definitions

All of the ciphers described in this document operate as modes of an underlying block cipher, meaning that the algorithms make calls to some underlying block cipher and base their security on the security of that underlying cipher. For the purposes of this specification, AES DUKPT is specified for use with Advanced Encryption Standard (AES). References for the definition of this algorithm can be found in the ASC X9 SD-34 registry.

Within the following algorithm descriptions, the following constant is used to specify the block size of the AES algorithm:

```
AES_BYTES = 16
```

The following AES operations are used in the function descriptions. The functions assume that AES is used with a 128-, 192- or 256-bit key:

— AES_Encrypt_ECB(K, Y): given a key K and a 16-byte value Y, return the AES ECB encryption of X as a 16-byte array.

— AES_Decrypt_ECB(K, Y): given a key K and a 16-byte value Y, return the AES ECB decryption of X as a 16-byte array.

5.4 Host-to-host UKPT

Host-to-host UKPT is similar to DUKPT, in that:

— the key used to protect the information in the message (e.g. PIN or data) is not pre-established;

— information needed by the receiving SCD to determine the correct transaction key is sent in the transaction message.

However, unlike DUKPT, the key(s) used in the transaction are not derived from a base key using a one-way function. The UKPT key is pre-established and for each transaction a non-secret transaction-specific seed value [including variant(s) for key separation if needed] is used to calculate the key(s) necessary to decrypt (as the first step to translation or verification).

The 16-byte seed value to be combined with the UKPT key may be randomly generated for each transaction by the SCD or may be a counter, depending upon implementation. Use of random seeds eliminates the need to store a counter and increment it for each key use. If the data is a counter, the host application supplies that counter to the encrypting SCD. If randomly generated, the SCD provides the seed together with the encrypted data in the response back to the host application for inclusion in the message to the receiving host.

If a counter is used for the seed value, a method to ensure the same seed value is not used for multiple transactions shall exist.

The message sent to the receiving host includes the seeds, which the receiving host SCD will need in order to calculate the correct transaction key(s) to perform the decryption or translation.

Verification shall be performed using the CMAC function as described in the ISO/IEC 9797 series.

All requirements related to a DUKPT BDK apply to the UKPT key.

Annex A
(informative)

Key and component check values

A.1 General

A check value is a cryptographic function of the key that is used to verify the correct key or component. The check value is not considered a secret parameter, which places significant security requirements on the check value algorithm and its usage.

A.2 Legacy approach (deprecated)

The previous edition of this document recommended computing the check value by encrypting an all-zero block using the key or component as the encryption key, see Figure A.1. When using this method, the check value available outside the SCD is the leftmost *n*-bits of the result, where *n* is at most 24 bits (six hexadecimal digits).

There are a number of known security problems with this approach. First, if the check value consists of the entire ciphertext, then it is possible that the check value can be used to enter a known ciphertext into the system. For example, the check value for a key wrapping key is identical to the ECB key wrap of a single-DEA key with a value of all zeros.

Even with a check value of only four or six digits, revealing the partial encryption of all zeros can cause security problems with some modes of operation, particularly Counter Mode and CMAC.

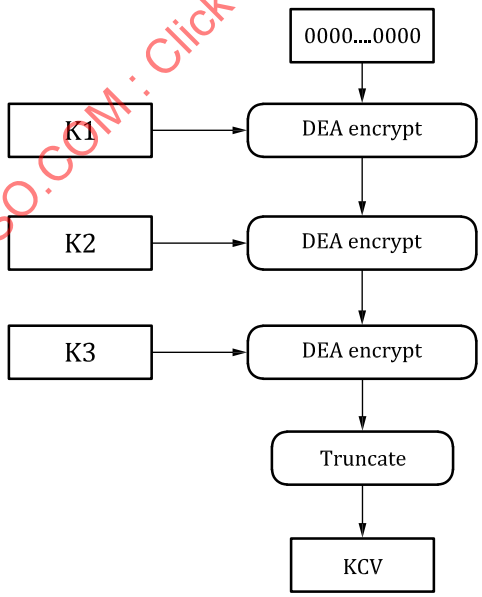


Figure A.1 — Legacy generation of key check value

A.3 CMAC-based check values

A.3.1 Algorithm

Check values calculated on an all-zero n-bit block, each bit a binary zero, using the CMAC algorithm as specified in the ISO/IEC 9798 series and where *n* is 128 for AES and 64 for TDEA.

The check value shall be the leftmost *k*-bits of the result, with a length consistent with 4.8.2.

The block cipher used in the CMAC function shall be the block cipher of the key itself.

The CMAC value is calculated according to Figure A.2.

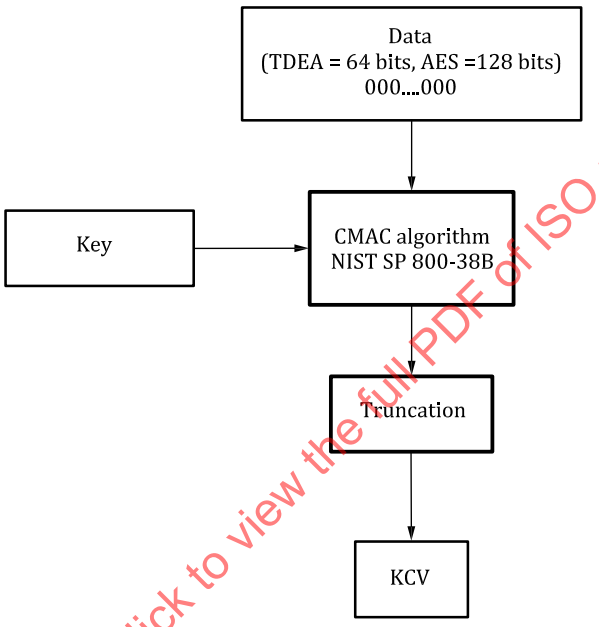


Figure A.2 — CMAC generation of key check value

A.3.2 Examples

Algorithm	Key	Check value
AES-128	1234567890ABCDEFFEDCBA0987654321	3F077B8FAA
AES-192	1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF	0C1589E01B
AES-256	000011112222333344445555666677778888999AAAABBBBCCCCD-DDDEEEEFFFF	F665910B72
TDEA	0123456789ABCDEF FEDCBA9876543210 (every 8th bit set for odd parity)	0A8245

A.4 Check values for HMAC keys

This algorithm should be used to generate and verify check values as shown in Figure A.3 with input values as shown in Table A.1.

VP = TruncN(SHA256(KA || KT || KL || K)) where:

VP	128-bit verification pattern
TruncN(x)	Truncation of the string x to the leftmost N bits Result shall be truncated to 128 bits at most

SHA256(x)	SHA-256 hash of the string x
KA	One-byte constant for the algorithm of key (HMAC 0x03)
KT	Two-byte constant for the type of key (MAC 0x0002)
KL	Two-byte bit length of the clear key value
K	Clear key value left-aligned and padded on the right with binary zeros to byte boundary
||	String concatenation

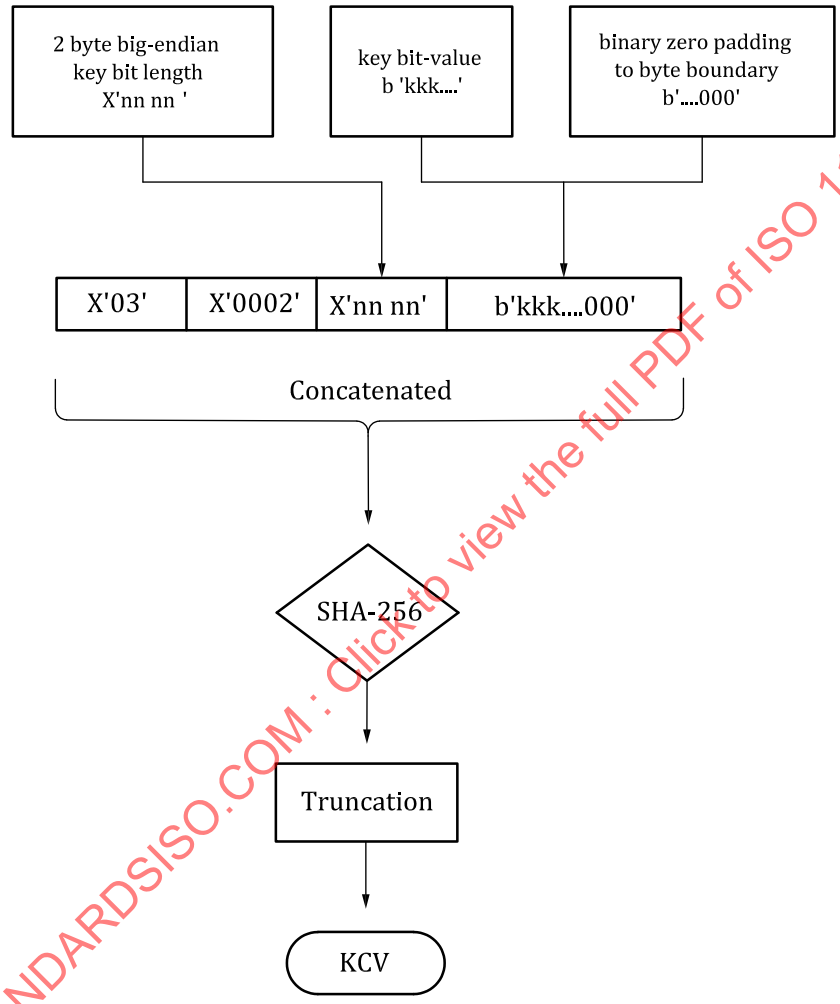


Figure A.3 — Key check value generation for HMAC keys

Sample calculations are shown in Table A.1.

Table A.1 — Input values for generating HMAC key check value

Key strength	Key	Encoded form	KCV
80 bit	0001020304050607080 9	0300020050000102030405060708 09	0d4ba604b022ae222d09786c567f677e
NOTE 1 All values are hexadecimal. NOTE 2 Encoded form has the appropriate pre-pend bytes, pre-pended.			

Table A.1 *(continued)*

Key strength	Key	Encoded form	KCV
128 bit	000102030405060708090A0B0C0D0E0F	03000200800001020304050607080 90A0B0C0D0E0F	fd726aa5b146a85cfdc8312bd989921c
256 bit	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	03000201000001020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	d54cae6d23ba82a1b44a228c22c6aefd
520 bit	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F40	03000202080001020304050607080 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F40	de278e848ef2be66942f4ab4e00c0e30
NOTE 1 All values are hexadecimal. NOTE 2 Encoded form has the appropriate pre-pend bytes, pre-pended.			

Annex B
(normative)

Split knowledge during transport

B.1 General

When two or more items need to remain separate during transport (when not in the possession of the authorized custodian, which is covered in 4.9.1.3.1) to support management of a key with split knowledge, it is necessary to implement a process to detect (and prevent where possible) any unauthorized access during the transportation and provide adequate evidence to demonstrate that no exposure occurred. Such items include but are not limited to:

a) components or shares of a key (e.g. printed or written on paper, on electronic media, in an HMD);

b) an SCD used to carry components or shares and the authentication parameters used to enable or access it;

c) physical keys that when used in concert enable an SCD to perform sensitive key management functions (e.g. brass keys).

This annex describes the required procedural elements and the evidence that shall be maintained in order to demonstrate split knowledge during transport.

B.2 Packaging

The packaging that protects the item during transit includes all barriers put around the item that are meant to protect it from disclosure. TEA bags are defined in 3.81.

Each item not contained within an SCD shall be contained in a sealed TEA bag. For written items:

a) It shall not be feasible to reveal the secret, even with manipulation of any interior barrier without opening the TEA bag (e.g. unfolding a trifold piece of paper or opening an unsealed envelope that are contained within the TEA bag).

b) The packaging shall be sufficient to ensure the value can't be read through the TEA bag. This can be a combination of a security envelope inside a TEA bag or could depend upon the TEA bag itself. A courier mailer is not sufficient to prevent disclosure.

For items shipped within transport media:

— Packaging shall be sufficient to protect items stored within transport media such that it cannot be read or accessed through the packaging.

— When selecting packaging for transport media, consideration should be given to the fact that it is possible to access various forms of media through certain types of packaging material without detection. For example, a magnetic stripe card needs to be protected from being read while remaining inside the TEA bag.

B.3 Separate shipment

Items shall be addressed using the name of the authorized custodian rather than the role (e.g. "Eve L. Mallory", not "Key Custodian 1").

Separation of the items shall be achieved by utilizing different communication channels. Examples include but are not limited to:

— component A shipped using courier company 1 and component B shipped using courier company 2;

— SCD shipped with courier company 1 and its associated password communicated via SMS.

When choosing the method with which to separate item shipment, consideration should be given to the impact if the key were to be compromised during transit (e.g. components for a key already protecting production information might warrant additional separation by date. A master file key already protecting other keys might warrant confirmation of receipt for each shipment before the next package is sent).

B.4 Receipt confirmation

Participation from the sender and receiver is necessary to ensure that the package received is the package that was sent, and that it was received intact. The process shall include:

a) designation of the individuals (not roles) authorized to send and receive items (i.e. “John Smith” not “Component Holder 1”);

b) use of a transportation channel(s) that allows for tracking (for items being shipped);

c) an inspection by the receiver for signs of tampering;

d) confirmation that the package was not substituted (e.g. the receiver providing the TEA bag number to be confirmed by the sender);

e) if an SCD, verification that it was not substituted using information that was received separate from the package itself (e.g. confirming the device serial number or a cryptographic means to validate the SCD was not substituted).

B.5 Transport logging

Logs detailing the activities for shipping and receiving shall be consistent with 4.1.4 and shall also include:

a) items transported;

b) method of transport;

c) evidence that the package was not substituted and showed no signs of tamper;

d) evidence indicating the device was not subject to substitution.

Annex C
(informative)

Trust models and key establishment

C.1 Introduction

Trust models and key establishment protocols are used in combination for key management. These methods consist of three trust models (three party, two party, prior trust) and three key establishment protocols (unilateral key transport, bilateral key transport, key agreement). Any of the trust models can be used with any of the key establishment protocols.

Mechanisms shall exist to establish trust for all public keys used in the key exchange and key transport protocols.

C.2 Trust models

C.2.1 Three-party model using third-party CA

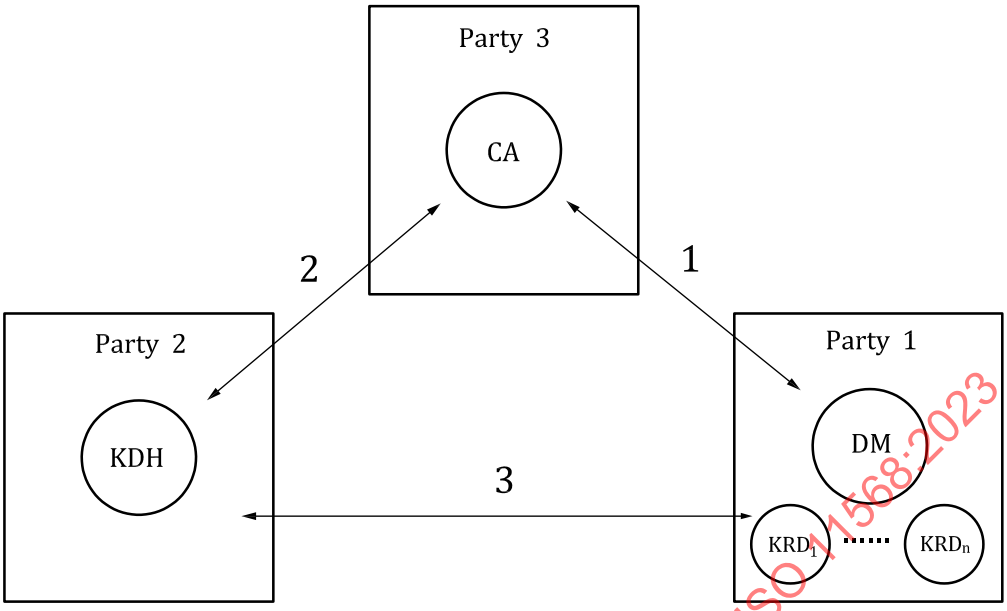
In a three-party trust model, each party of a communicating pair will use a third party to establish the trust for their public keys. This third party is a certificate authority (CA). Trust is placed on the CA to verify the identity and integrity of the requestor and the requestor's public key. Once validated, the CA issues a digital certificate for the requestor's public key (signed by the CA's private key). Parties that have the issuing CA's public key can authenticate the certificate as legitimately issued by the CA.

All public keys used in this model are validated by chaining their validity back to the third-party CA's root key pair. The CA does not actively participate in the key transport protocol, but merely participates in establishing the foundation that forms the trust relationship for the exchange of the public keys.

The CA's public key is the root key of trust in the key hierarchy. The CA's root public and private key pair shall be generated and managed in accordance with this document. The CA shall operate in a secure environment as described in the ISO 13491 series that provides secure processing to ensure the integrity of the public key infrastructure environment. The CA's public key shall be distributed as per the appropriate subclause of ISO 21188.

The CA shall have PKI practices and policies in accordance with ISO 20038. Such policies establish the mechanisms and provide assurance for validating the authentic identification of the certificate requestors. Valid identification is essential in establishing the trust of the public keys in this system.

In Figure C.1, party 1 is a third-party CA, party 2 is an organization operating a key distribution host (KDH) and party 3 is a manufacturer of the key receiving device (KRD), which is typically a PED. The manufacturer is signing the KRD public key. Trust exists between the CA and the device manufacturer, so that as device public keys are signed by the manufacturer's key (which in turn has been signed by the CA) the chain of trust back to the CA is established. Trust also exists between the CA and KDH to enable mutual authentication of the KDH and KRD during key distribution process.



Key

DM device manufacturer

Figure C.1 — Three-party example: key distribution to KRDs

C.2.2 Two-party model – one party acts as CA

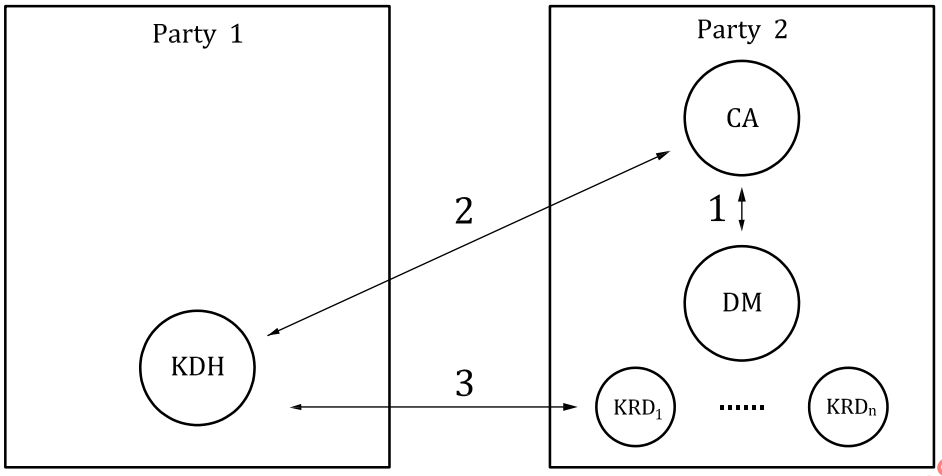
In this scenario, one party assumes a role similar to that of the three-party CA in that all public keys used in this model are validated by chaining their validity back to one of the party's root key pair. The party assuming the role of the CA (CA party) has a root verification key pair. The other party generates a public–private key pair and, via an integrity-assured channel, sends its public key to the first party to be digitally signed by the CA party's root verification private key (this is done once in an initialization stage.) The CA party generates a public–private key pair for encryption or decipherment and sends the public encryption key digitally signed by its own verification-signing key (self-signed) to the second party. Then, in the course of the key transport protocol, all public keys exchanged are validated using the CA party's root public verification key.

While this explanation addresses two parties, this method can also be used in a one-to-many entity relationship, with one party assuming the role similar to that of the CA party. For example, a KRD manufacturer (party acting as CA) could sign a multiplicity of KRD public keys (same party) and also sign the KDH public key (second party).

The first party's public key (i.e. the CA party's public key) is the root key of trust in this scenario. This verification key pair will assume the role of the CA root key pair and therefore shall be generated and managed in accordance with 4.6.2. The CA party shall operate in a secure environment as described in the ISO 13491 series that provides secure processing to ensure the integrity of the public key infrastructure environment. The CA party's public key shall be distributed as per the appropriate subclause of ISO 21188.

The CA party shall have PKI practices and policies in accordance with ISO 21188. Such policies establish the mechanisms and provide assurance for validating the authentic identification of the certificate requestors. Valid identification is essential in establishing the trust of the public keys in this system.

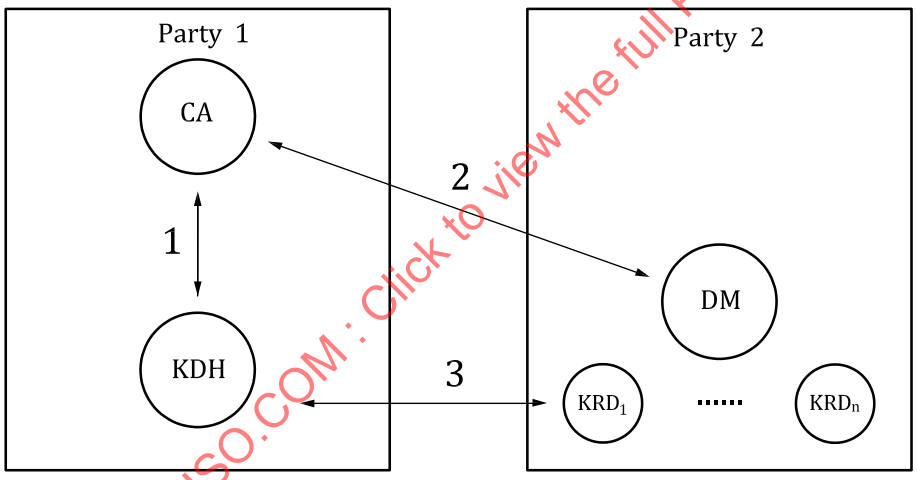
Examples can be found in Figures C.2, C.3 and C.4.



Key
DM device manufacturer

NOTE The manufacturer of KRDs is CA.

Figure C.2 — Two-party example: key distribution to KRDs



Key
DM device manufacturer

NOTE KDH is CA.

Figure C.3 — Two-party example: key distribution to KRDs

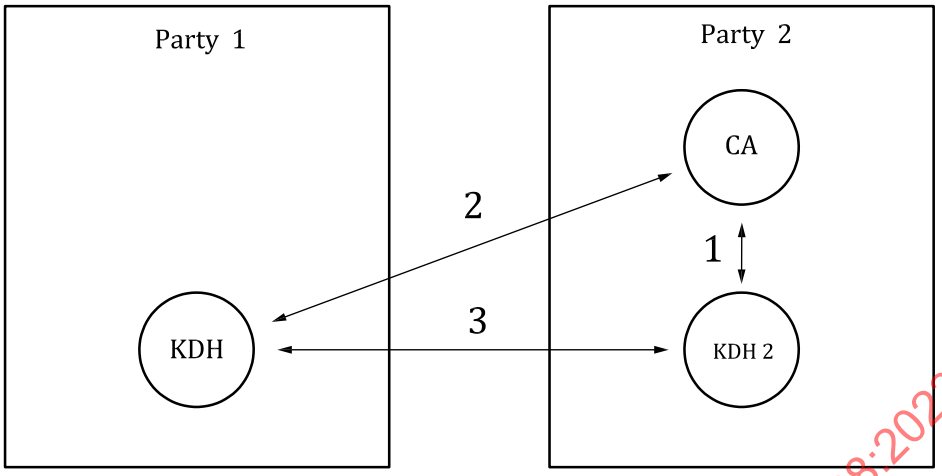


Figure C.4 — Two-party example: key distribution between host nodes

C.2.3 Prior trust model – no CA

In circumstances where use of a CA is impractical or inappropriate, methods other than use of a CA may be used to establish trust in the public keys used (e.g. manually exchanging media containing the public keys).

There shall be a prequalification process to ensure the integrity of the keying material and the authenticity of the request and requestor. The parties establishing manual trust shall have policies and procedures in place consistent with ISO 21188 for identification and authentication purposes. This model shall not be used for key exchange to PEDs (since certificate revocation lists are not applicable and it would be impractical to notify all affected parties in a timely manner in case of private key compromise).

The prior trust model can be used where two institutions want to exchange symmetric keys on a host-to-host communications link without need for the use of a CA. The institutions would first exchange public keys using an out-of-band process that mutually validates the authenticity of the parties involved and ensures that valid public keys are received. Once the public keys are exchanged, one of the protocols listed in C.3 may be employed.

Examples for use of this model include, but are not limited to:

— exchange of symmetric keys between vendors to facilitate manufacture of an SCD;

— exchange of symmetric keys between processors and financial institutions and/or third-party service providers;

— exchange of symmetric keys within organisational internal systems. (e.g. between business units).

C.3 Symmetric key establishment protocols

C.3.1 General

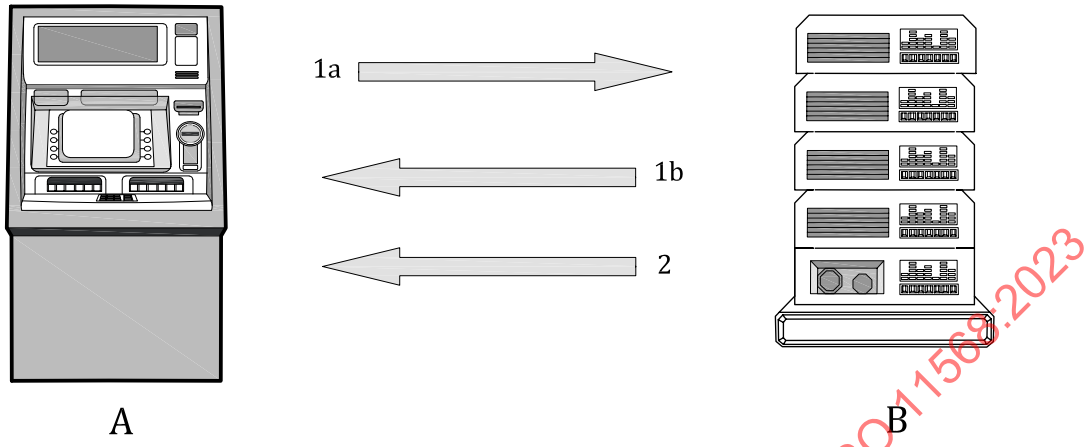
This subclause describes the approved asymmetric key methodologies for distributing symmetric keys to devices. Each method shall use one of the trust models described in C.2 to establish trust between the communicating pair. These key establishment methods fall into two categories:

— key transport protocol (unilateral key transport or bilateral key transport);

— key agreement protocol.

C.3.2 Unilateral key transport method

Figure C.5 represents a high-level overview of the steps that occur where a KDH generates the symmetric key and transports it to the KRD.



Key

A KRD

B KDH

Figure C.5 — High-level overview of key transport method (unilateral)

After the KDH begins communication, the high-level steps involved in the key transport protocol method (unilateral key transport where the KDH generates and sends the symmetric key) are as follows:

a) If public keys have not previously been exchanged and stored:

   1) the KDH sends its verification public key to the KRD;

   2) the KRD sends its public key(s) to the KDH;

   3) each key recipient validates the received public key, for example by validating a certificate.

b) The KDH generates a symmetric key – unique (except by chance) for this KRD – inside the KDH's SCD and retains it (e.g. stores it in its local database as a cryptogram encrypted under its SCD's symmetric master key). The KDH's SCD also encrypts the symmetric key under the KRD's public encryption key. Observe that the symmetric key does not exist anywhere in the clear (except within the KDH's SCD). The KDH signs the cryptogram with the KDH's signature key and sends it to the KRD.

   The KDH sends the cryptogram of the symmetric key encrypted under the KRD's public encryption key to the KRD. Only the KRD will be able to decrypt the symmetric key because only the KRD has the corresponding private decipherment key.

c) The KRD verifies the signature using the KDH's verification key and decrypts the symmetric key using the KRD's private decipherment key.

d) The KDH and the KRD now share a symmetric key.

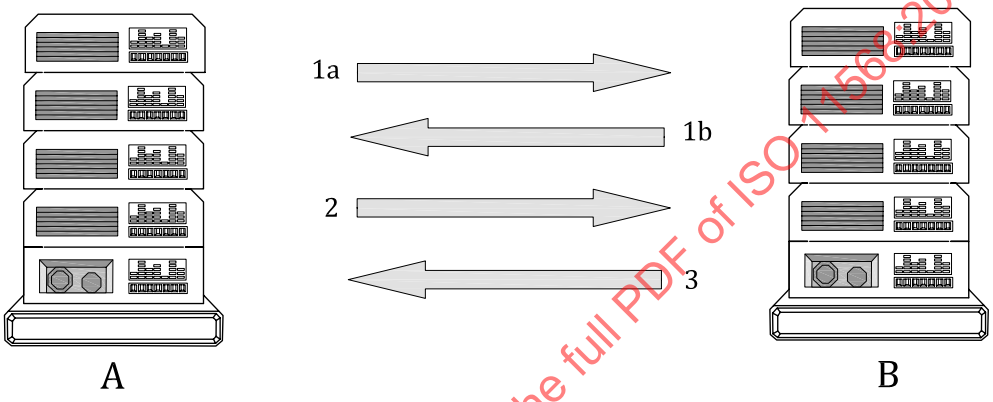
Many other steps can occur for the actual protocol. Figure C.5 does not contain the protocol messages. It does not show any of the detailed steps for the generation of the public and private key pairs, any involvement of a CA to issue certificates or any of the details of digital signature techniques that can be used to prevent a middleperson attack on this system. It does not show the use of the KDH's or KRD's signature or verification keys. It is meant to show only the very simplest steps in the process, not the actual protocol.

This method is based on the ISO/IEC 11770 series. Implementations shall conform to the requirements stated in the ISO/IEC 11770 series, which reference the RSA asymmetric algorithm.

Unauthorized modification, substitution or replay of the transported symmetric key and middleperson attacks shall be prevented or detected.

C.3.3 Bilateral key transport method (both parties generate and share symmetric key – joint control)

Another form of the key transport method is for bilateral key transport where both the initiator and the responder each generate a symmetric key within SCDs and send each key to the other party. Figure C.6 represents a high-level overview of the steps that occur. Two key distribution hosts (i.e. no PEDs) are depicted in this example, denoted KDH1 and KDH2.



Key

A KDH1

B KDH2

Figure C.6 — High-level overview of key transport method (bilateral)

After KDH1 begins communication, the high-level steps involved in the key transport protocol method (bilateral key transport where the KDH1 and the KDH2 each generate a symmetric key and send the symmetric key to the other party) are as follows:

a) If public keys have not previously been exchanged and stored:

   1) KDH1 sends its public keys (verification and encryption) to KDH2;

   2) KDH2 sends its public keys (verification and encryption) to KDH1.

b) KDH1 generates a symmetric key – unique (except by chance) for KDH2 – inside KDH1's SCD and retains it (e.g. stores it in its local database as a cryptogram encrypted under its SCD's symmetric master key). KDH1's SCD also encrypts the symmetric key under KDH2's public encryption key. Observe that KDH1's symmetric key does not exist anywhere in the clear (except within KDH1's SCD). KDH1 signs the cryptogram with KDH1's signature key.

   KDH1 sends the cryptogram of KDH1's symmetric key encrypted under KDH2's public encryption key to KDH2. Only KDH2 will be able to decrypt the symmetric key because only KDH2 has the corresponding private decipherment key.

c) KDH2 verifies the signature using KDH1's verification key and decrypts KDH1's symmetric key within its SCD using KDH2's private decipherment key.

   KDH2 generates a symmetric key – unique for KDH1 (except by chance) – inside KDH2's SCD and retains it to combine it with KDH1's contribution for the final symmetric key as explained in step 5. KDH2's SCD also encrypts the symmetric key under KDH1's public encryption key. Observe that

KDH2's symmetric key does not exist anywhere in the clear (except within KDH2's SCD). KDH2 signs the cryptogram with KDH2's signature key.

KDH2 sends the cryptogram of KDH2's symmetric key encrypted under KDH1's public encryption key to KDH1. Only KDH1 will be able to decrypt the key component because only KDH1 has the corresponding private decipherment key.

d) KDH1 verifies the signature using KDH2's verification key and decrypts KDH2's symmetric key within its SCD using KDH1's private decipherment key.

e) KDH2 and KDH1 now share symmetric keys. If these symmetric keys are to be combined to form a shared secret key, then a one-way function shall be used.

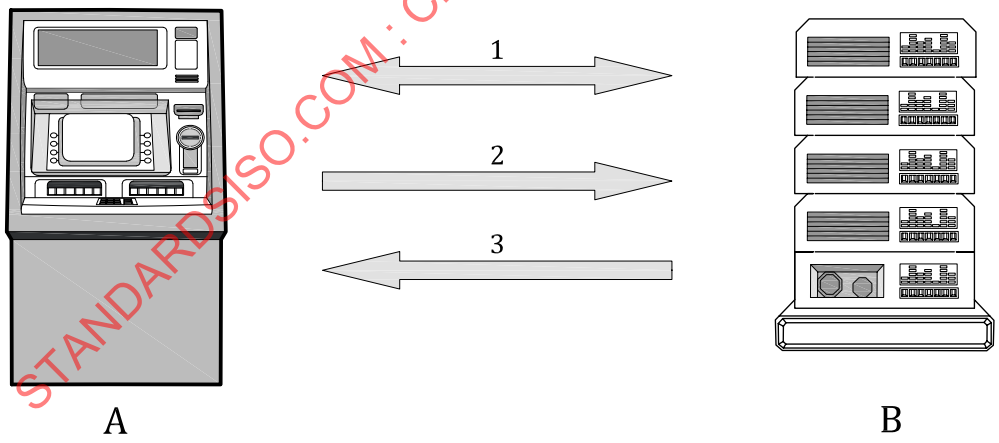
Many other steps can occur for the actual protocol. Figure C.6 does not contain the protocol messages. It does not show any of the detailed steps for the generation of the public and private key pairs, or any involvement of a CA to issue certificates or any of the details of digital signature techniques that can be used to prevent a middleperson attack on this system. It does not show the use of the responder's or initiator's signature or verification keys. It is meant to show only the very simplest steps in the process, not the actual protocol.

This method is based on the ISO/IEC 11770 series. Implementations shall conform to the requirements stated in the ISO/IEC 11770 series, which references the RSA asymmetric algorithm or ANSI X9.63 which is ECC.

Unauthorized modification, substitution or replay of the transported symmetric key and middleperson attacks shall be prevented or detected.

C.3.4 Key agreement method

Figure C.7 illustrates how the Diffie-Hellman algorithm is used for key agreement. In this method, both the host and interfacing device exchange information that is then used to derive the symmetric key. In a key agreement scheme, both parties contribute information that is used by both to derive a shared secret key (k = k' in the following steps).



Key

A ATM

B host

Figure C.7 — High-level overview of key agreement method

The high-level steps involved in the key agreement protocol method are as follows:

a) The device and the host authenticate each other and then agree upon two shared values (g and n). This may be done across an open channel. Details of the rules for generating g and n are not discussed here, but g and n do not have to be secret.

b) The device generates within its SCD a private random large integer x and calculates a public value (X) to send to the host. The public value (X) is equal to g^x mod n.

c) The host generates within its SCD a private random large integer y and calculates a public value (Y) to send to the device. The public value (Y) is equal to g^y mod n.

d) The device calculates a key k equal to Y^x mod n within its SCD.

e) The host calculates a key k' equal to X^y mod n within its SCD.

f) Observe that k and k' are equal to each other but are independently calculated and never have to be transported from one party to the other.

g) The host and this device now share the same key and can now establish a PIN encrypting key.

Many other steps can occur. Figure C.7 does not show any of the detailed steps for the generation of g and n or the special mathematical relationships they have, the public (X and Y) and private (x and y) key pairs or any involvement of a CA to issue certificates or any of the details of digital signature techniques that can be used to prevent a middleperson attack on this system.

Other key agreement protocols (e.g. MQV model) may be used but are not detailed in this document. Only ANSI- or ISO/IEC-approved key agreement protocols shall be used.

Table C.1 identifies the acceptable combinations.

Table C.1 — Trust models and key establishment protocols

		Trust models		
		Three party	Two party	Prior trust
Key establishment protocol	Unilateral key transport	✓	✓	✓
	Bilateral key transport	✓	✓	✓
	Key agreement	✓	✓	✓

Annex D
(informative)

Symmetric key life cycle

The states that make up a key's lifetime are collectively referred to as the key's life cycle. Termination and archiving should consider all places in which the key exists. An operation that changes a key's state is referred to as a life cycle operation. The key life cycle consists of three phases:

a) Pre-use: during which the key is generated and optionally stored prior to its use.

b) Use: during which the key is distributed among communicating parties for operational use.

   In a process where both communicating parties contribute to the generation of a new key, key generation and distribution are closely integrated.

   Some key management schemes are designed for transforming keys automatically during operational use.

c) Post-use: during which a key is archived or terminated.

   A key is considered to be a single object of which multiple instances can exist at different locations and in different forms. A clear distinction is made between the following operations:

   — destruction of a single key instance;

   — deletion of a key from a given location, which implies destruction of all instances of this key at that location;

   — termination of a key, which implies deletion of the key from all locations.

   Archiving requires the destruction of all instances of the key except the instance(s) archived.

Figure D.1 gives a schematic overview of the key life cycle. It shows how a given operation on a key changes its state.

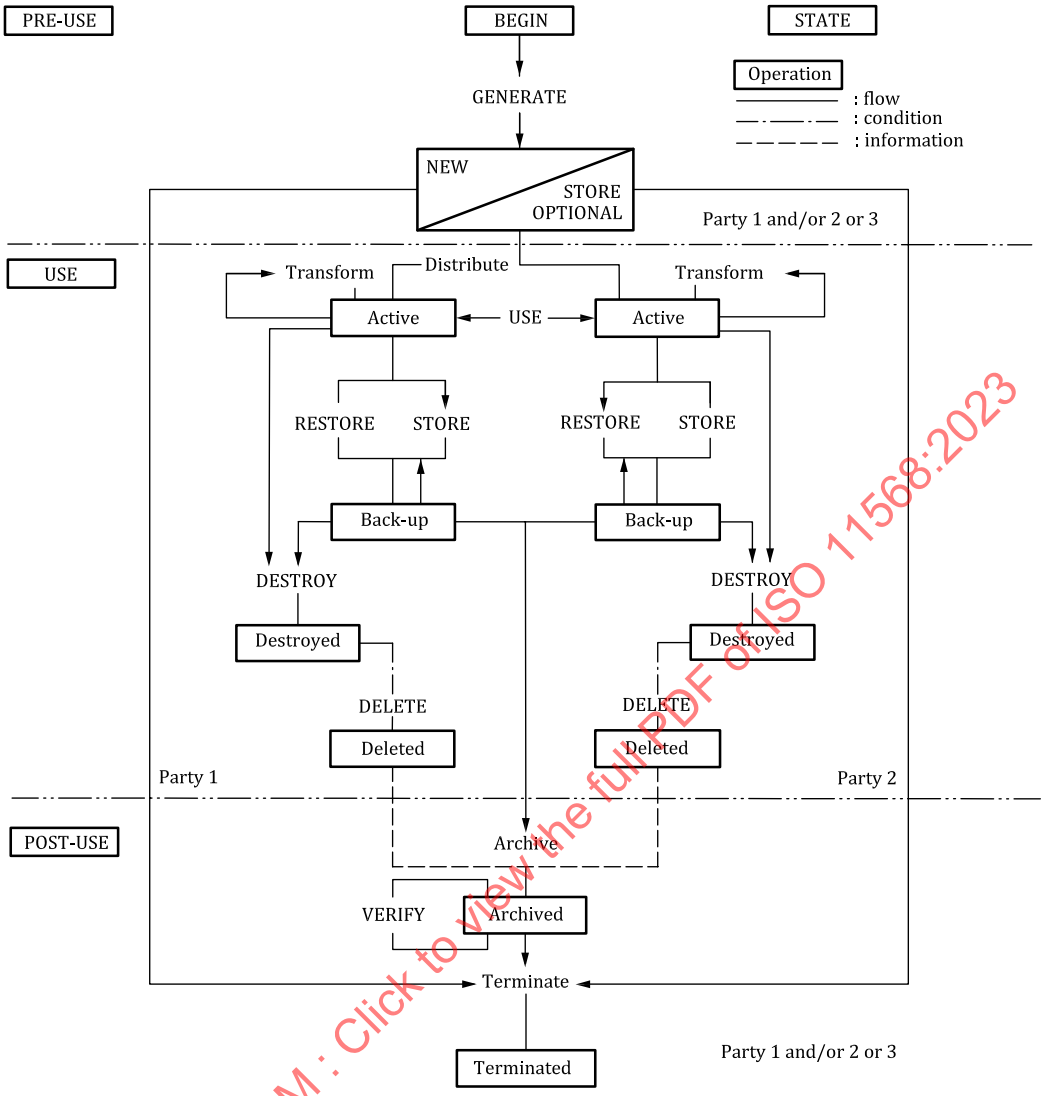


Figure D.1 — Symmetric key life cycle schematic

Annex E
(informative)

Asymmetric key life cycle phases

The key life cycle consists of three phases:

a) pre-use: during which the key pair is generated and might be transferred;

b) use: during which the public key is distributed to one or more parties for operational use and the private key is retained in an SCD;

c) post-use: during which the public key of a key pair is archived and the private key of a key pair is terminated.

A schematic overview of the private key (S) life cycle and the public key (P) life cycle are given respectively in Figures E.1 and E.2. The figures show how a given operation on a key changes its state. The life cycle of public key certificates can be found in ISO 21188.

A cryptographic key is considered to be a single object of which multiple instances may exist at different locations and in different forms. A clear distinction is made between the following operations:

— distribution of the public key to a communicating party;

— transfer of a key pair to its owner in an implementation where the party does not have the capacity to generate key pairs.

Figures E.1 and E.2 depict the asymmetric key life cycle and public key life cycle, respectively.

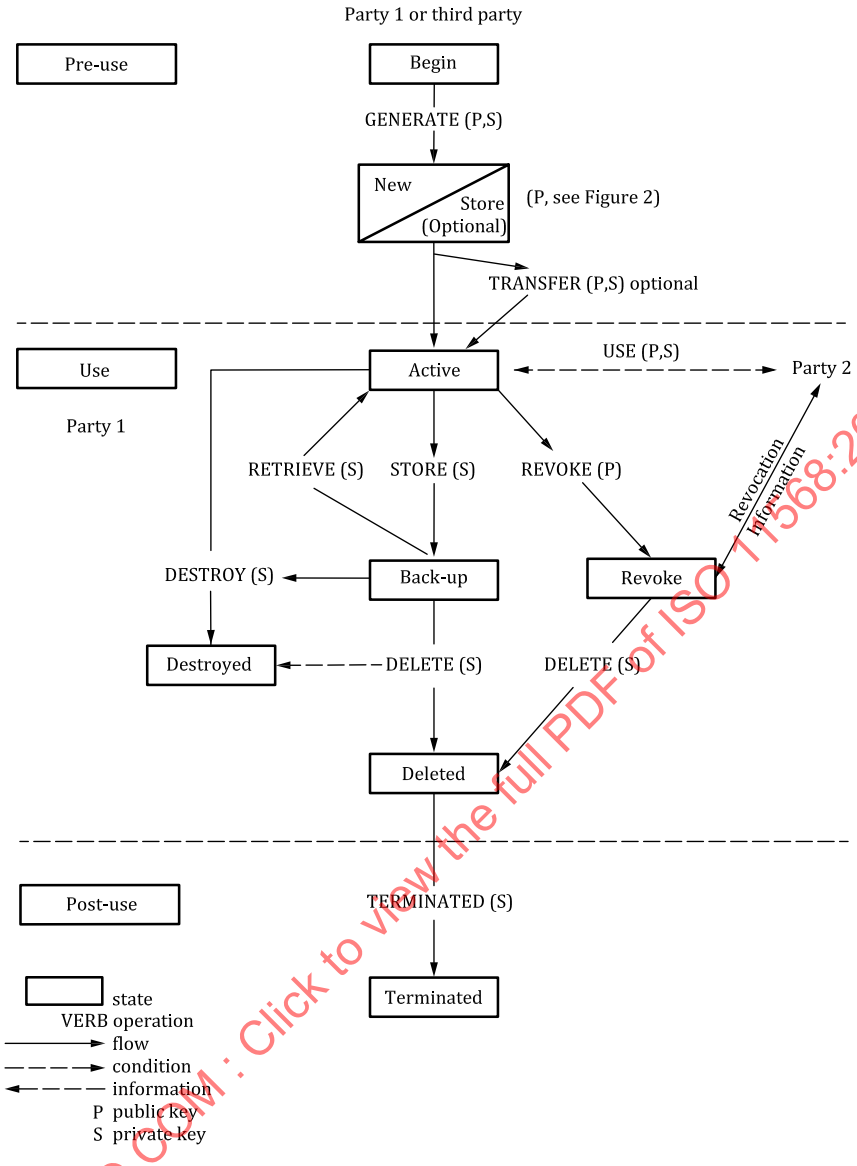


Figure E.1 — Asymmetric key life cycle schematic

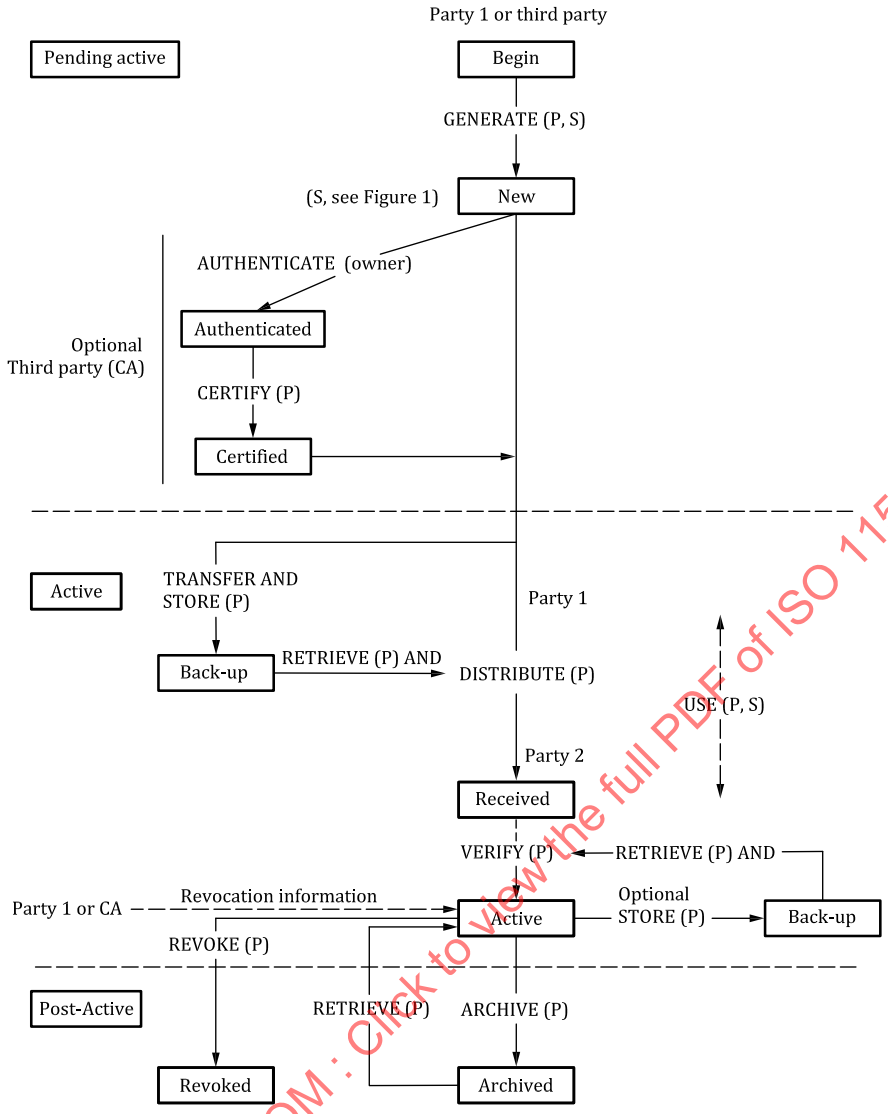


Figure E.2 — Public key life cycle

Annex F
(normative)

Approved algorithms

F.1 General

This annex identifies the algorithms that are approved for use in key management using public key cryptography.

F.2 Approved algorithms

F.2.1 Algorithms approved for public key transport systems

— RSA-based encryption methods as defined in ISO/IEC 18033-2;

— ECIES-KEM (key encapsulation method) as defined in ISO/IEC 18033-2.

F.2.2 Algorithms approved for public key agreement systems

— Diffie-Hellman;

— EC-DH;

— EC-DHE;

— EC-MQV;

— Blinded EC-DH.

NOTE Algorithms for public key agreement systems are described in the ISO/IEC 11770 series.

F.2.3 Algorithms approved for digital signatures

— RSA, see ISO/IEC 9796-2;

— DSA, see ISO/IEC 14888-3;

— ECDSA, see ISO/IEC 14888-3;

— EC-SDSA, see ISO/IEC 14888-3.

F.2.4 Approved hash functions

Approved hash functions are those described in ISO/IEC 10118.

Annex G
(informative)

AES DUKPT pseudocode notation

G.1 General

The functions and subroutines in Annex H are described using a simplified pseudocode programming language. The descriptive pseudocode is intended to describe the algorithms as directly and unambiguously as possible. Implementations of AES DUKPT on real systems will involve optimizations and details (e.g. memory management) that may cause them to look quite different than the pseudocode version detailed here.

G.2 Notation syntax

G.2.1 General

Pseudocode is distinguished from descriptive text by the use of the `Courier` font. The pseudocode in this document describes a set of functions that consists of a list of statements that operate on values. The following subclauses describe how values, statements and functions are written.

G.2.2 Values

There are three types of value in the pseudocode: strings, Booleans and integer.

— Integers are scalar, unsigned whole quantities with no upper limit on size. Integer variables can also be assigned the special non-numeric value ERROR.

— Booleans can take two defined values, TRUE and FALSE. The special values INVALID and VALID are treated as identical to FALSE and TRUE, respectively.

— Strings are ordered arrays of integers.

The following operators for string variables are used:

— Length(X) returns (as an integer) the number of integers contained in the string.

— Concat(X, Y) returns a string that contains the integers contained in X, followed by the integers contained in Y.

— X[i], where i is a integer, denotes the ith value of the array X, with the first element referenced by X[0].

— X[i..j], where i and j are integers, with i < j, denotes the substring starting at the ith element, and ending with the jth element.

The following set of numeric operations are used. The result of any of these operators with any argument set to ERROR will return the value ERROR.

— ceil(x) returns the smallest integer value equal to or larger than x.

— floor(x) returns the largest integer value equal to or smaller than x.

— log(x, y) returns the base y logarithm of x.

— ln2(x) returns the base 2 logarithm of x.

— x+y, x-y, x*y return the sum, difference or product of x and y, respectively.

— X++ increments x by 1 and returns the new value.

— x mod y returns the remainder of x divided by y.

— x^y returns x raised to the power of y.

— x XOR y returns the exclusive-or of x and y.

— x AND y returns the bitwise and of x and y.

— x OR y returns the bitwise or of x and y.

— x << y returns the result of a logical shift of x to the left by y bits.

— x >> y returns the result of logical shift of x to the right by y bits.

The following operators for Boolean values are allowed:

— a && b returns TRUE if a and b are TRUE and FALSE otherwise.

— a || b returns TRUE if a or b are TRUE, and FALSE otherwise.

The following operators take integer values and return Boolean values:

— a == b returns TRUE if a and b are identical values and FALSE otherwise.

— a != b returns TRUE if a and b are different and FALSE otherwise.

G.2.3 Statements

The pseudocode defines computational actions by statements. Statements take one of the following forms:

Compound statements. Statements can be grouped with curly brackets as follows:

{ <statement1> <statement2> ... <statement n> }

Variable assignment. Statements of the form:

<name> = *<value>*

assign the specified value to a variable identified by <name>. By convention, names starting with capital letters are assigned string values and names starting with lower-case letters are assigned integer values.

Conditionals. Statements of the form:

`if (`*<Boolean value>*`)` *<statement 1>* `else` *<statement 2>*

evaluate the Boolean value. If TRUE, <statement 1> is executed; otherwise, <statement 2> is executed. The else clause is optional and can be left off.

Loops. Statements of the form:

`for`(*<statement 1>;<Boolean value>;<statement 2>*) *<statement 3>*

do the following actions. First, execute <statement 1>. Next, evaluate the <Boolean value>. If <Boolean value> is TRUE, execute <statement 2> then execute <statement 3>. Repeat this process until <Boolean value> is FALSE; in this case, do not execute either <statement 2> or <statement 3> but continue with the next line of the pseudocode.

Statements of the form:

`while`*(<Boolean value>) <statement 1>*

evaluate <Boolean value> and, if TRUE, execute <statement 1> then repeat this action.

Returns. Statements of the form:

`return` *<value>;*

exit from a function and set the value of the function to <value>.

G.2.4 Functions

Functions are sequences of pseudocode statements that take a set of variable parameters and return an integer, Boolean or string. Functions are defined by the following syntax:

<name>(<parameter 1>, …. , <parameter n>)

{

<statement 1>

….

<statement n>

}

Functions are invoked as values of the form:

<name>(<value 1>,…,<value n>)

The value of a function is defined by the return statement executed within the function.

Annex H
(informative)

AES DUKPT test vectors

AES DUKPT test vectors for validating an implementation can be found at https://x9.org/standards/x9-24-part-3-test-vectors/. The counter values are chosen to cover the first eight transactions, the first "skipped bit" near transaction 131072, a transaction chosen at random from the middle of the space (8675309), the last five transaction counters and the DUKPT update key encryption key (counter = 0xFFFFFFFF).

The following important use cases are explicitly covered:

— generating 128-bit AES keys from a 128-bit AES BDK/Initial Key;

— generating 128-bit AES keys from a 256-bit AES BDK/Initial Key;

— generating 256-bit AES keys from a 256-bit AES BDK/Initial Key;

— generating 2-key TDEA keys from a 128-bit AES BDK/Initial Key;

— generating 3-key TDEA keys from a 128-bit AES BDK/Initial Key.

Examples of the calculation of AES PIN blocks (Format 4) are also given.

Finally, a trace of all the internal calculations for the derivation of the first eight transaction counters is given, both for the host and terminal sides of the algorithm. The inputs and outputs of each function from the pseudocode, as well as the values of all internal variables, are given. This is done to aid in debugging implementations of this document, to make it easy to spot where a given implementation diverges from correct behaviour.

Annex I
(informative)

TDEA-derived unique key per transaction[2)]

I.1 Use

This information is provided for compatibility with existing implementations that were based on versions of ANSI X9.24-1 issued in 2008 and earlier. This description is functionally identical to previous descriptions. For new implementations, please refer to the method described in the main body of this document.

This annex describes the TDEA DUKPT key management method for PIN entry devices. The technique as described supports double-length TDEA keys and not triple-length TDEA keys. In addition to deriving a unique PIN encryption key for each transaction, the method includes the optional derivation of unique-per-transaction MAC keys and data encryption keys.

Initially, this annex describes the various preferred PIN entry device storage areas and then specifies the preferred PIN entry device functions that are used for the TDEA method of DUKPT of processing PINs. Either the methodology as described or its functional equivalent is performed to ensure that the Key Serial Number and encrypted PIN block are generated correctly.

In any of the following descriptions, the bit or byte order is such that the leftmost bit, decimal digit, hexadecimal digit or byte is the most significant and the rightmost bit, decimal digit, hexadecimal digit or byte is the least significant. In the case of the Shift Register, this implies that bit #1 is the most significant and bit #21 is the least significant bit.

This annex makes use of the term “initial key”, which is equivalent to the term “initial DUKPT key”.

I.2 Storage areas

I.2.1 General

The PIN entry device maintains certain storage areas only during the PIN-processing operation. Other storage areas are permanently maintained.

I.2.2 PIN processing

The contents of the following storage area relating to PIN processing are maintained only during a given PIN encryption operation.

Account number register (12 decimal digits): The account number register holds the 12 rightmost digits (excluding the check digit) of the Primary Account Number received from the terminal in the "Request PIN Entry" command.

I.2.3 Key management

The following storage areas relating to key management are maintained for the life of the PIN entry device, beginning with the time of the "Load Initial Key" command:

Initial Key Serial Number register (59 bits): The IKSN register holds the leftmost 59 bits of the Key Serial Number that is initially injected into the PIN entry device along with the initial key loaded during

2) This annex reproduces the original TDEA DUKPT implementation as described in ANSI X9.24-1-2009, Annex A.

the "Load Initial Key" command. The contents of this register remain fixed for the service-life of the PIN entry device or until another "Load Initial Key" command is issued. For details regarding the Key Serial Number, see Figure I.1.

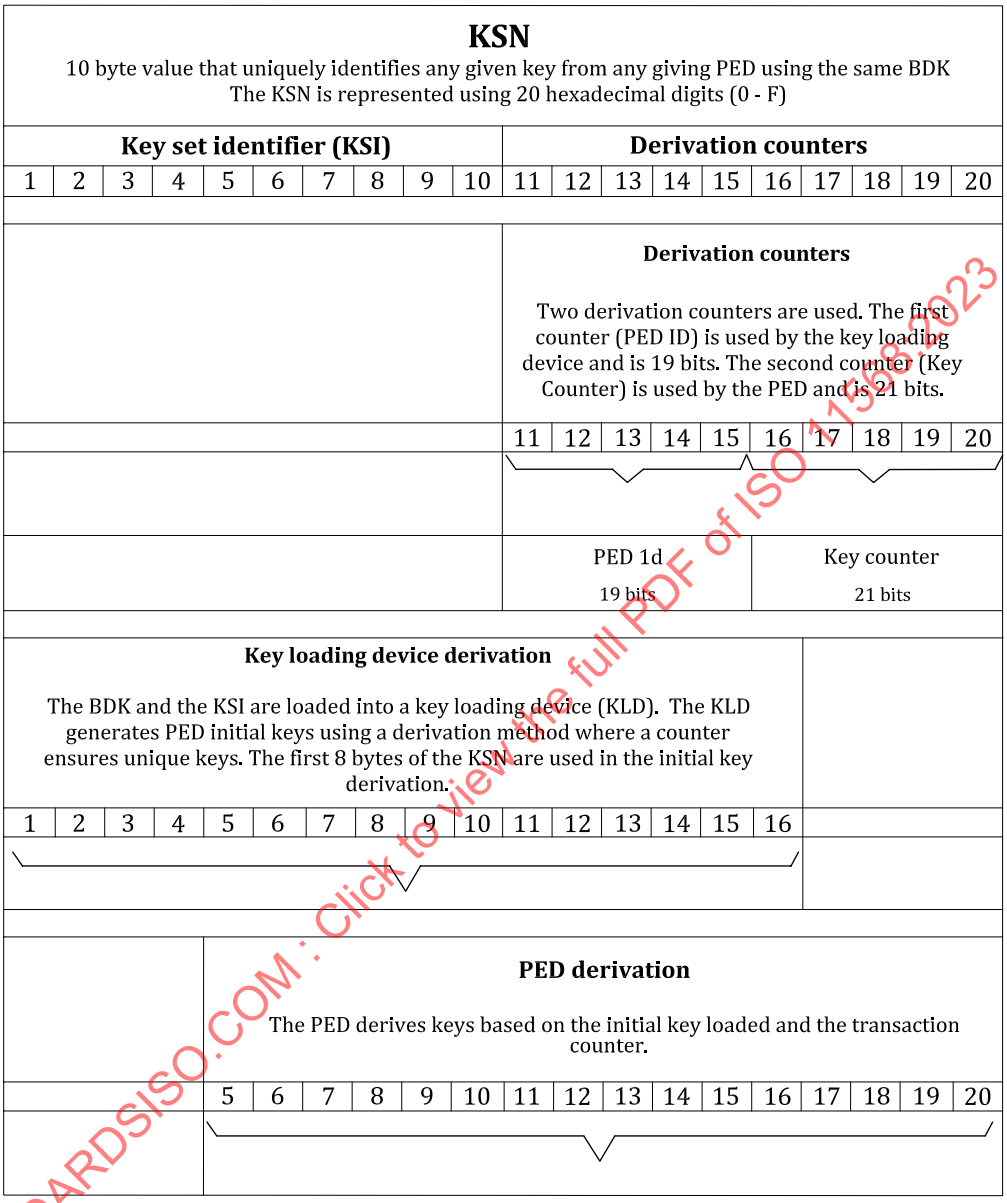


Figure I.1 — Key Serial Number details

Encryption Counter (21 bits): A counter of the number of PIN encryptions that have occurred since the PIN entry device was first initialized. Certain counter values are skipped (as explained below), so that over one million PIN encryption operations are possible.

NOTE The concatenation (left to right) of the Initial Key Serial Number Register and the Encryption Counter form the 80-bit (20 hexadecimal digits) Key Serial Number Register.

Intermediate Derivation Key Registers (21 registers of 34 hexadecimal digits each): A set of 21 registers, numbered #1 to #21, that are used to store future PIN encryption keys. Each register includes an indication as to whether the register contains a key or is empty. In this annex, this is accomplished using a longitudinal redundancy check (LRC).

The following storage areas relating to key management are required on a temporary basis and may be used for other purposes by other PIN processing routines:

Current Key Pointer (approximately four hexadecimal digits): Current Key Pointer contains the address of the Intermediate Derivation Key Register whose contents are being used in the current cryptographic operation. **![Current Key Pointer]** identifies the contents of the Future Key Register (FKR) whose address is contained in the Current Key Pointer.

Shift Register (21 bits): A 21-bit register, whose bits are numbered from left to right as #1 to #21. This register contains 20 "zero" bits and a single "one" bit. This register is used to select one of the Intermediate Derivation Key Registers. The Intermediate Derivation Key Register to be selected is the register that corresponds to the bit in the Shift Register that contains the single "one" bit.

Crypto Register-1 (16 hexadecimal digits): A register used in performing cryptographic operations.

Crypto Register-2 (16 hexadecimal digits): A second register used in performing cryptographic operations.

Key Register (32 hexadecimal digits): A register used to hold a cryptographic key.

MAC Key Register (32 hexadecimal digits): An optional register used to hold a cryptographic key for performing message authentication.

MAC Response Key Register (32 hexadecimal digits): An optional register used to hold a cryptographic key for performing message authentication.

Data Encryption Key Register (32 hexadecimal digits): An optional register used to hold a cryptographic key for performing data encryption.

Data Encryption Response Key Register (32 hexadecimal digits): An optional register used to hold a cryptographic key for performing data encryption.

I.3 Processing algorithms

I.3.1 General

The PIN entry device may receive any of three different commands from the device to which it is attached. These commands are: "Load Initial Key", "Request PIN Entry", and "Cancel PIN Entry". The processing steps to be performed in each case are indicated below.

Whenever the text indicates that the LRC is used, either an LRC or a CRC may be used.

I.3.2 Load key

I.3.2.1 “Load Initial Key" (external command)

a) Store the initial key, as received in the externally initiated command, into Intermediate Derivation Key Register #21.

b) Generate and store the LRC on this Intermediate Derivation Key Register.

c) Write the address of Intermediate Derivation Key Register #21 into the Current Key Pointer.

d) Store the Key Serial Number, as received in the externally initiated command, into the Key Serial Number Register. (This register is the concatenation of the Initial Key Serial Number Register and the Encryption Counter.)

e) Clear the Encryption Counter (the 21 rightmost bits of the Key Serial Number Register).

f) Set bit #1 (the leftmost bit) of the Shift Register to "one", setting all of the other bits to "zero".

g) Go to "New Key-3".

I.3.2.2 "New Key" (local label)

a) Count the number of "one" bits in the 21-bit Encryption Counter. If this number is less than 10, go to "New Key-1".

b) Erase the key at ![Current Key Pointer].

c) Set the LRC for ![Current Key Pointer] to an invalid value (e.g. increment the LRC by one).

d) Add the Shift Register to the Encryption Counter. (This procedure skips those counter values that would have more than 10 "one" bits.)

e) Go to "New Key-2".

I.3.2.3 "New Key-1" (local label)

a) Shift the Shift Register right one bit (end-off). (A "zero" is shifted into position #1, the leftmost bit of the register.)

b) If the Shift Register now contains all zeros (i.e. the single "one" was shifted off), go to "New Key-4", else go to “New Key-3”.

I.3.2.4 "New Key-3" (local label)

a) The Shift Register, right justified in 64 bits, padded to the left with zeros, OR'ed with the 64 rightmost bits of the Key Serial Number Register, is transferred into Crypto Register-1.

b) Copy ![Current Key Pointer] into the Key Register.

c) Call the subroutine “Derivation Process” in D.4.6.

d) Store the contents of Crypto Register-1 into the left half of the Intermediate Derivation Key Register indicated by the position of the single "one" bit in the Shift Register.

e) Store the contents of Crypto Register-2 into the right half of the Intermediate Derivation Key Register indicated by the position of the single "one" bit in the Shift Register.

f) Generate and store the LRC on this Intermediate Derivation Key Register.

g) Go to "New Key-1".

I.3.2.5 "New Key-4" (local label)

a) Erase the key at ![Current Key Pointer].

b) Set the LRC for ![Current Key Pointer] to an invalid value (e.g. increment LRC by one).

c) Add one to the Encryption Counter.

d) Go to “New Key-2”.

I.3.2.6 "New Key-2" (local label)

If the Encryption Counter contains all zeros, cease operation. (The PIN entry device is now inoperative, having encrypted more than 1 million PINs.) If not all zeros, go to “Exit”.

I.3.2.7 "Exit" (local label)

Return to original calling routine. (Processing of the current externally initiated command is completed, and the PIN entry device is ready for the next command. The Current Key Pointer, Account Number Register, Shift Register, and Crypto Pointer may now be used for other purposes.)