INTERNATIONAL STANDARD

ISO/IEC 11770-3

Third edition
2015-08-01

Information technology — Security techniques — Key management —

Part 3:
Mechanisms using asymmetric techniques

Technologies de l'information — Techniques de sécurité — Gestion de clés —

Partie 3: Mécanismes utilisant des techniques asymétriques

Reference number
ISO/IEC 11770-3:2015(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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 document 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 and IEC 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 on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 27, *Security techniques*.

This third edition cancels and replaces the second edition (ISO/IEC 11770-3:2008 with ISO/IEC 11770-3/Cor1:2009), which has been technically revised.

ISO/IEC 11770 consists of the following parts, under the general title *Information technology — Security techniques — Key management*:

— *Part 1: Framework*

— *Part 2: Mechanisms using symmetric techniques*

— *Part 3: Mechanisms using asymmetric techniques*

— *Part 4: Mechanisms based on weak secrets*

— *Part 5: Group key management*

— *Part 6: Key derivation*

Further parts may follow.

Introduction

This part of ISO/IEC 11770 describes schemes that can be used for key agreement and schemes that can be used for key transport.

Public key cryptosystems were first proposed in the seminal paper by Diffie and Hellman in 1976. The security of many such cryptosystems is based on the presumed intractability of solving the discrete logarithm problem over certain finite fields. Other public key cryptosystems such as RSA are based on the difficulty of the integer factorization problem.

A third class of public key cryptosystems is based on elliptic curves. The security of such a public key system depends on the difficulty of determining discrete logarithms in the group of points of an elliptic curve. When based on a carefully chosen elliptic curve, this problem is, with current knowledge, much harder than the factorization of integers or the computation of discrete logarithms in a finite field of comparable size. All known general purpose algorithms for determining elliptic curve discrete logarithms take exponential time. Thus, it is possible for elliptic curve based public key systems to use much shorter parameters than the RSA system or the classical discrete logarithm based systems that make use of the multiplicative group of some finite field. This yields significantly shorter digital signatures, as well as system parameters, and allows for computations using smaller integers.

This part of ISO/IEC 11770 includes mechanisms based on the following:

— finite fields;

— elliptic curves;

— bilinear pairings.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this International Standard may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from those in Annex H.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

Information technology — Security techniques — Key management —

Part 3:
Mechanisms using asymmetric techniques

1 Scope

This part of ISO/IEC 11770 defines key management mechanisms based on asymmetric cryptographic techniques. It specifically addresses the use of asymmetric techniques to achieve the following goals.

a) Establish a shared secret key for use in a symmetric cryptographic technique between two entities *A* and *B* by key agreement. In a secret key agreement mechanism, the secret key is computed as the result of a data exchange between the two entities *A* and *B*. Neither of them should be able to predetermine the value of the shared secret key.

b) Establish a shared secret key for use in a symmetric cryptographic technique between two entities *A* and *B* via key transport. In a secret key transport mechanism, the secret key is chosen by one entity *A* and is transferred to another entity *B*, suitably protected by asymmetric techniques.

c) Make an entity's public key available to other entities via key transport. In a public key transport mechanism, the public key of entity *A* shall be transferred to other entities in an authenticated way, but not requiring secrecy.

Some of the mechanisms of this part of ISO/IEC 11770 are based on the corresponding authentication mechanisms in ISO/IEC 9798-3.[6]

This part of ISO/IEC 11770 does not cover certain aspects of key management, such as

— key lifecycle management,

— mechanisms to generate or validate asymmetric key pairs, and

— mechanisms to store, archive, delete, destroy, etc. keys.

While this part of ISO/IEC 11770 does not explicitly cover the distribution of an entity's private key (of an asymmetric key pair) from a trusted third party to a requesting entity, the key transport mechanisms described can be used to achieve this. A private key can in all cases be distributed with these mechanisms where an existing, non-compromised key already exists. However, in practice the distribution of private keys is usually a manual process that relies on technological means such as smart cards, etc.

This part of ISO/IEC 11770 does not specify the transformations used in the key management mechanisms.

NOTE To provide origin authentication for key management messages, it is possible to make provisions for authenticity within the key establishment protocol or to use a public key signature system to sign the key exchange messages.

2 Normative references

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

ISO/IEC 10118 (all parts), *Information technology — Security techniques — Hash-functions*

ISO/IEC 11770-1, *Information technology — Security techniques — Key management — Part 1: Framework*

ISO/IEC 15946-1, *Information technology — Security techniques — Cryptographic techniques based on elliptic curves — Part 1: General*

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

3 Terms and definitions

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

3.1
asymmetric cryptographic technique
cryptographic technique that uses two related transformations, a public transformation (defined by the public key) and a private transformation (defined by the private key), and has the property that given the public transformation, then it is computationally infeasible to derive the private transformation

Note 1 to entry: A system based on asymmetric cryptographic techniques can either be an encryption system, a signature system, a combined encryption and signature system, or a key agreement scheme. With asymmetric cryptographic techniques there are four elementary transformations: *signature* and *verification* for signature systems, *encryption* and *decryption* for encryption systems. The signature and the decryption transformations are kept private by the owning entity, whereas the corresponding verification and encryption transformations are published. There exist asymmetric cryptosystems (e.g. RSA) where the four elementary functions can be achieved by only two transformations: one private transformation suffices for both signing and decrypting messages, and one public transformation suffices for both verifying and encrypting messages. However, since this does not conform to the principle of key separation, throughout this part of ISO/IEC 11770 the four elementary transformations and the corresponding keys are kept separate.

3.2
asymmetric encryption system
system based on asymmetric cryptographic techniques whose public transformation is used for encryption and whose private transformation is used for decryption

3.3
asymmetric key pair
pair of related keys where the private key defines the private transformation and the public key defines the public transformation

3.4
certification authority
CA
centre trusted to create and assign public key certificates

3.5
collision-resistant hash-function
hash-function satisfying the following property: it is computationally infeasible to find any two distinct inputs which map to the same output

[SOURCE: ISO/IEC 10118-1:2000, 3.2]

3.6
decryption
reversal of a corresponding encryption

[SOURCE: ISO/IEC 11770-1:2010, 2.6]

3.7
digital signature
data unit appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to verify the origin and integrity of the data unit and protect the sender and the recipient of the data unit against forgery by third parties, and the sender against forgery by the recipient

3.8
distinguishing identifier
information which unambiguously distinguishes an entity

[SOURCE: ISO/IEC 11770-1:2010, 2.9]

3.9
encryption
(reversible) transformation of data by a cryptographic algorithm to produce ciphertext, i.e. to hide the information content of the data

[SOURCE: ISO/IEC 11770-1:2010, 2.10]

3.10
entity authentication
corroboration that an entity is the one claimed

[SOURCE: ISO/IEC 9798-1:2010, 3.14]

3.11
entity authentication of entity *A* to entity *B*
assurance of the identity of entity *A* for entity *B*

3.12
explicit key authentication from entity *A* to entity *B*
assurance for entity *B* that entity *A* is the only other entity that is in possession of the correct key

Note 1 to entry: Implicit key authentication from entity *A* to entity *B* and key confirmation from entity *A* to entity *B* together imply explicit key authentication from entity *A* to entity *B*.

3.13
forward secrecy with respect to entity *A*
property that knowledge of entity *A*'s long-term private key subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys

3.14
forward secrecy with respect to both entity *A* and entity *B* individually
property that knowledge of entity *A*'s long-term private key or knowledge of entity *B*'s long-term private key subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys

Note 1 to entry: This differs from mutual forward secrecy in which knowledge of both entity *A*'s and entity *B*'s long-term private keys do not enable recomputation of previously derived keys.

3.15
hash-function
function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:

— it is computationally infeasible to find for a given output, an input which maps to this output;

— it is computationally infeasible to find for a given input, a second input which maps to the same output

Note 1 to entry: Computational feasibility depends on the specific security requirements and environment.

Note 2 to entry: For the purposes of this standard all hash-functions are assumed to be collision-resistant (see 3.5).

[SOURCE: ISO/IEC 10118-1:2000, 3.5]

3.16
implicit key authentication from entity *A* to entity *B*
assurance for entity *B* that entity *A* is the only other entity that can possibly be in possession of the correct key

3.17
key
sequence of symbols that controls the operation of a cryptographic transformation (e.g. encryption, decryption, cryptographic check function computation, signature calculation, or signature verification)

[SOURCE: ISO/IEC 11770-1:2010, 2.12]

3.18
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.

3.19
key commitment
process of committing to use specific keys in the operation of a key agreement scheme before revealing the specified keys

3.20
key confirmation from entity *A* to entity *B*
assurance for entity *B* that entity *A* is in possession of the correct key

3.21
key control
ability to choose the key or the parameters used in the key computation

3.22
key derivation function
function that outputs one or more shared secrets, for use as keys, given shared secrets and other mutually known parameters as input

3.23
key establishment
process of making available a shared secret key to one or more entities, where the process includes key agreement and key transport

3.24
key token
key management message sent from one entity to another entity during the execution of a key management mechanism

3.25
key transport
process of transferring a key from one entity to another entity, suitably protected

3.26
message authentication code
MAC

string of bits which is the output of a MAC algorithm

Note 1 to entry: A MAC is sometimes called a cryptographic check value (see for example ISO 7498-2[1]).

[SOURCE: ISO/IEC 9797-1:2011, 3.9]

3.27
Message Authentication Code algorithm
MAC algorithm
algorithm for computing a function which maps strings of bits and a secret key to fixed-length strings of bits, satisfying the following two properties:

— for any key and any input string, the function can be computed efficiently;

— for any fixed key, and given no prior knowledge of the key, it is computationally infeasible to compute the function value on any new input string, even given knowledge of a set of input strings and corresponding function values, where the value of the ith input string might have been chosen after observing the value of the first $i - 1$ function values (for integers $i > 1$)

Note 1 to entry: A MAC algorithm is sometimes called a cryptographic check function (see for example ISO 7498-2[1]).

Note 2 to entry: Computational feasibility depends on the user's specific security requirements and environment.

[SOURCE: ISO/IEC 9797-1:2011, 3.10]

3.28
mutual entity authentication
entity authentication which provides both entities with assurance of each other's identity

3.29
mutual forward secrecy
property that knowledge of both entity *A*'s and entity *B*'s long-term private keys subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys

3.30
one-way function
function with the property that it is easy to compute the output for a given input but it is computationally infeasible to find an input which maps to a given output

3.31
prefix free representation
representation of a data element for which concatenation with any other data does not produce a valid representation

3.32
private key
key of an entity's asymmetric key pair that is kept private

Note 1 to entry: The security of an asymmetric system depends on the privacy of this key.

[SOURCE: ISO/IEC 11770-1:2010, 2.35]

3.33
public key
key of an entity's asymmetric key pair which can usually be made public without compromising security

Note 1 to entry: In the case of an asymmetric signature system, the public key defines the verification transformation. In the case of an asymmetric encryption system, the public key defines the encryption transformation, conditional on the inclusion of randomisation elements. A key that is "publicly known" is not necessarily globally available. The key can only be available to all members of a pre-specified group.

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

3.34
public key certificate
public key information of an entity signed by the certification authority and thereby rendered unforgeable

3.35
public key information
information containing at least the entity's distinguishing identifier and public key, but can include other static information regarding the certification authority, the entity, restrictions on key usage, the validity period, or the involved algorithms

3.36
secret key
key used with symmetric cryptographic techniques by a specified set of entities

3.37
sequence number
time variant parameter whose value is taken from a specified sequence which is non-repeating within a certain time period

[SOURCE: ISO/IEC 11770-1:2010, 2.44]

3.38
signature system
system based on asymmetric cryptographic techniques whose private transformation is used for signing and whose public transformation is used for verification

3.39
third party forward secrecy
property that knowledge of a third party's private key subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys

Note 1 to entry: Instead of third party forward secrecy, master key forward secrecy is also used in Reference [19].

3.40
time stamp
data item which denotes a point in time with respect to a common time reference

3.41
time-stamping authority
trusted third party trusted to provide a time-stamping service

[SOURCE: ISO/IEC 13888-1:2009, 3.58]

3.42
time variant parameter
data item used to verify that a message is not a replay, such as a random number, a time stamp or a sequence number

Note 1 to entry: If a random number is used. then this is as a challenge in a challenge-response protocol. See also ISO/IEC 9798-1:2010, Annex B.

[SOURCE: ISO/IEC 9798-1:2010, 3.36]

3.43
trusted third party
security authority or its agent, trusted by other entities with respect to security related activities

[SOURCE: ISO/IEC 9798-1:2010, 3.38]

4 Symbols and abbreviations

The following symbols and abbreviations are used in this part of ISO/IEC 11770.

A, *B*, *C*	distinguishing identifiers of entities
BE	encrypted data block
BS	signed data block
CA	certification authority
$Cert_A$	entity *A*'s public key certificate
D_A	entity *A*'s private decryption transformation function
d_A	entity *A*'s private decryption key
E	elliptic curve, either given by an equation of the form $Y^2 = X^3 + aX + b$ over the field GF(p^m) for p>3 and a positive integer *m*, by an equation of the form $Y^2 + XY = X^3 + aX^2 + b$ over the field GF(2^m), or by an equation of the form $Y^2 = X^3 + aX^2 + b$ over the field GF(3^m), together with an extra point O_E referred to as the point at infinity, which is denoted by *E*/GF(p^m), *E*/GF(2^m), or *E*/GF(3^m), respectively
E_A	entity *A*'s public encryption transformation function
e_A	entity *A*'s public encryption key
F	key agreement function
F(*h*,*g*)	key agreement function using as input a factor *h* and a common element *g*
FP	key agreement function based on pairing
G	point on *E* with order *n*
g	common element shared publicly by all the entities that use the key agreement function F
gcd(*a*,*b*)	greatest common divisor of two integers *a* and *b*
GF(p^m), GF(2^m), GF(3^m)	finite field with p^m , 2^m, 3^m elements for a prime p>3 and a positive integer *m*
h_A	entity *A*'s private key agreement key
hash	hash-function
j	cofactor used in performing cofactor multiplication
K	secret key for a symmetric cryptosystem
K_{AB}	secret key shared between entities *A* and *B*

NOTE 1 In practical implementations the shared secret key should be subject to further processing before it can be used for a symmetric cryptosystem.

kdf	key derivation function
KT	key token
KT_A	entity *A*'s key token

KT_{Ai}	key token sent by entity A after processing phase i
l	supplementary value used in performing cofactor multiplication
M	data message
MAC	Message Authentication Code
$\mathrm{MAC}_K(Z)$	output of a MAC algorithm when using as input the secret key K and an arbitrary data string Z
MQV	Menezes-Qu-Vanstone
n	prime divisor of the order (or cardinality) of an elliptic curve E over a finite field
O_E	elliptic curve point at infinity
P	point on an elliptic curve E
p_A	entity A's public key-agreement key
pairing	pairing defined over an elliptic curve and used in FP
parameters	parameters used in the key derivation function
PKI_A	entity A's public key information
P_X	public key-agreement key in an elliptic curve of entity X
q	prime power p^m for some prime $p \neq 3$ and some integer $m \geq 1$
r	random number generated in the course of a mechanism
r_A	random number issued by entity A in a key agreement mechanism
S_1, S_2, S_3	sets of elements
S_A	entity A's private signature transformation function
s_A	entity A's private signature key
T	trusted third party
Text$i$	i th optional text, data or other information that may be included in a data block, if desired
TVP	time-variant parameter such as a random number, a time stamp, or a sequence number
V_A	entity A's public verification transformation function
v_A	entity A's public verification key
w	one-way function
$\mathrm{X}(P)$	x-coordinate of a point P
$\sqrt{q}$	square root of a positive number q
$\#E$	order (or cardinality) of an elliptic curve E

$\|\|$	concatenation of two data elements
$\lceil x \rceil$	smallest integer greater than or equal to the real number x
$\sum$	digital signature
$\pi(P)$	$(\mathrm{X}(P) \bmod 2^{\lceil \rho/2 \rceil}) + 2^{\lceil \rho/2 \rceil}$ where $\rho = \lceil \log_2 n \rceil$ and X(P) is the x-coordinate of the point P

NOTE 2 No assumption is made on the nature of the signature transformation. In the case of a signature system with message recovery, $S_A(M)$ denotes the signature $\sum$ itself. In the case of a signature system with appendix, $S_A(M)$ denotes the message M together with the signature $\sum$.

NOTE 3 The keys of an asymmetric cryptosystem are denoted by lower case letters (indicating its function) indexed with the identifier of its owner, e.g., the public verification key of entity A is denoted by v_A. The corresponding transformations are denoted by upper case letters indexed with the identifier of their owner, e.g., the public verification transformation of entity A is denoted by V_A.

5 Requirements

It is assumed that the entities involved in a mechanism are aware of each other's claimed identities. This may be achieved by the inclusion of identifiers in information exchanged between the two entities, or it may be apparent from the context of use of the mechanism. Verifying the identity means checking that a received identifier field agrees with some known (trusted) or expected value.

If a public key is registered with an entity, then that entity shall make sure that the entity who registers the key is in possession of the corresponding private key (see ISO/IEC 11770-1 for further guidance on key registration).

6 Key derivation functions

The use of a shared secret as derived in Clause 10 as a key for a symmetric cryptosystem without further processing is not recommended. It will often be the case that the form of a shared secret established as a result of using a mechanism specified in this part of ISO/IEC 11770 will not conform to the form needed for a specific cryptographic algorithm, so some processing will be needed. Moreover, the shared secret (often) has arithmetic properties and relationships that might result in a shared symmetric key not being chosen from the full key space. It is therefore advisable to pass the shared secret through a key derivation function, e.g. involving the use of a hash function. The use of an inadequate key derivation function could compromise the security of the key agreement scheme with which it is used. It is recommended to use a one-way function as a key derivation function.

A key derivation function produces keys that are computationally indistinguishable from randomly generated keys. The key derivation function takes as input a shared secret and a set of key derivation parameters and produces an output of the desired length.

In order for the two parties in a key establishment mechanism to agree on a common secret key, the key derivation function shall be agreed upon (see ISO/IEC 11770-6 for further guidance on key derivation functions).

Annex C provides examples of key derivation functions.

7 Cofactor multiplication

This clause applies only to mechanisms using elliptic curve cryptography. The key agreement mechanisms in Clause 11 and the key transport mechanisms in Clauses 12 and 13 require that the user's private key or key token be combined with another entity's public key or key token. If the other entity's public key or key token is not valid (i.e., it is not a point on the elliptic curve, or is not in the subgroup of order n), then performing this operation may result in some bits of the private key being leaked to an attacker. One example of such an attack is known as the 'small subgroup attack'.

NOTE 1 The small subgroup attack is described in [37].

In order to prevent the 'small subgroup attack' and similar attacks, one option is to validate public keys and key tokens received from the other party using public key validation, as specified in ISO/IEC 11770-1.

As an alternative to public key validation, a technique called cofactor multiplication as specified in Clause 11 can be used. The values j and l, defined below, are used in cofactor multiplication.

If cofactor multiplication is used, there are two options:

— If compatibility with entities not using cofactor multiplication is not required, then let $j = \#E / n$ and $l = 1$. If this option is chosen, both parties involved shall agree to use this option, otherwise the mechanism will not work.

— If compatibility with entities not using cofactor multiplication is required, then let $j = \#E / n$ and $l = j^{-1} \bmod n$.

NOTE 2 The value $j^{-1} \bmod n$ will always exist since n is required to be greater than $4\sqrt{q}$ and therefore $\gcd(n, j) = 1$.

If cofactor multiplication is not required, then let $j = l = 1$.

Regardless of whether or not cofactor multiplication is used, if the shared key (or a component of the shared key) evaluates to the point at infinity (O_E), then the user shall assume that the key agreement procedure has failed.

It is particularly appropriate to perform public key validation or cofactor multiplication in the following cases:

— if the entity's public key is not authenticated;

— if the key token is not authenticated;

— if the user's public key is intended for a long-term use.

If the other entity's public key is authenticated and the cofactor is small, then the amount of information that can be leaked is limited. Thus, it may not always be necessary to perform these tests.

8 Key commitment

Clause 11 describes key agreement mechanisms in which the established key is the result of applying a one-way function to the private key-agreement keys. However, one entity may know the other entity's public key or key token prior to choosing their private key. As a result, such an entity can control the value of s bits in the established key, at the cost of generating 2^s candidate values for their private key-agreement key in the time interval between discovering the other entity's public key or key token and choosing their own private key.[31]

One way to address this concern (if it is a concern) at the cost of one additional message/pass in the protocol is through the use of key commitment. Key commitment can be performed by having the first entity hash the public key or key token and send the hash-code to the second entity; the second entity then replies with its public key or key token, and the first entity replies with its public key or key token. The second entity can now hash it and verify that the result is equal to the hash-code sent earlier.

9 Key confirmation

Explicit key confirmation is the process of adding additional messages to a key establishment protocol providing implicit key authentication, so that explicit key authentication and entity authentication are provided. Explicit key confirmation can be added to any method that does not possess it inherently. Key confirmation is typically provided by exchanging a value that can (with very high probablity) only be calculated correctly if the key establishment calculations were successful. Key confirmation from entity *A* to entity *B* is provided by entity *A* calculating a value and sending it to entity *B* for confirmation of entity *A*'s correct calculation. If mutual key confirmation is desired, then each entity sends a different value to the other.

Key confirmation is often provided by subsequent use of an established key, and if something is wrong then it is immediately detected. This is called implicit key confirmation. Explicit key confirmation in this case may be unnecessary. If one entity is not online (for example, in one-pass protocols used in store and forward (email) scenarios), then it is simply not possible for the other entity to obtain key confirmation. However, sometimes a key is established yet used only later (if at all), or the entity performing the key establishment process may simply not know if the resulting key will be used immediately or not. In these cases, it is often desirable to use a method of explicit key confirmation, as it may otherwise be too late to correct an error once detected. Explicit key confirmation can also be seen as a way of "firming up" security properties during the key establishment process and may be warranted if a conservative protocol design is deemed appropriate.

An example method of providing key confirmation using a MAC is as follows:

Entities *A* and *B* first perform one of the key establishment procedures specified in Clauses 11 and 12 of this part of ISO/IEC 11770. As a result, they expect to share a secret MAC key K_{AB}. They then perform the following procedure.

— Entity *B* forms the message *M*, an octet string consisting of the message identifier octet 0x02, entity *B*'s identifier, entity *A*'s identifier, the octet string KT_B corresponding to entity *B*'s key token (omitted if not present), the octet string KT_A corresponding to entity *A*'s key token (omitted if not present), the octet string p_B corresponding to entity *B*'s public key-establishment key (omitted if not present), the octet string p_A corresponding to entity *A*'s public key-establishment key (omitted if not present) and, if present, optional additional Text1, i.e.:

$M = 02||B||A||KT_B||KT_A||p_B||p_A||$Text1, where 0x02 is the message number.

— Entity *B* calculates $K_B = \text{kdf}(K_{AB})$, and then calculates $\text{MAC}_{KB}(M)$ for the message *M* under the (supposedly) shared secret key K_B for an appropriate MAC scheme.

— Entity *B* sends the message *M* and $\text{MAC}_{KB}(M)$ to entity *A*.

— Entity *A* calculates $K_A = \text{kdf}(K_{AB})$, computes $\text{MAC}_{KA}(M)$ using the received message *M*, and verifies $\text{MAC}_{KB}(M) = \text{MAC}_{KA}(M)$.

— Assuming the MAC verifies, entity *A* has received key confirmation from entity *B* (that is, entity *A* knows that K_A equals K_B). If mutual key confirmation is desired, entity *A* continues the protocol and forms the message *M'* as the octet string consisting of the message identifier octet 0x03, entity *A*'s identifier, entity *B*'s identifier, the octet string KT_A corresponding to entity *A*'s key token (omitted if not present), the octet string KT_B corresponding to entity *B*'s key token (omitted if not present), the octet string p_A corresponding to entity *A*'s public key-establishment key (omitted if not present), the octet string p_B corresponding to entity *B*'s public key-establishment key (omitted if not present) and optional additional octet string Text2, i.e.:

$M' = 03||A||B||KT_A||KT_B||p_A||p_B||$Text2, where 0x03 is the message number.

— Entity *A* calculates $\text{MAC}_{KA}(M')$ under the (supposedly) shared secret K_A using an appropriate MAC scheme.

— Entity *A* sends *M'* and $\text{MAC}_{KA}(M')$ to entity *B*.

— Entity B uses K_B to verify $\mathrm{MAC}_{K_A}(M')$ on the message M'. Assuming the MAC verifies, entity B has received key confirmation from entity A (that is, entity B knows that K_A equals K_B).

Other methods of key confirmation are possible. If the shared secret is to be used for data confidentiality (encryption), one entity can send the encryption of some specific plaintext known to the other entity, for example a block of all binary zeros or all binary ones. Care should be taken that any subsequent use of the key is very unlikely to encrypt the same plaintext as was used for key confirmation.

10 Framework for key management

10.1 General

This clause contains a high-level description of a framework for the key establishment mechanisms specified in this part of ISO/IEC 11770. Four categories of mechanism are defined (key agreement between two parties, key agreement between three parties, secret key transport and public key transport), together with requirements for their use.

10.2 Key agreement between two parties

This clause applies to the Key Agreement mechanisms 11.1 through 11.11 that describe key agreement between two parties. Key Agreement between two parties is the process of establishing a shared secret key between two entities A and B in such a way that neither of them can predetermine the value of the shared secret key. Key agreement mechanisms may provide for implicit key authentication; in the context of key establishment, implicit key authentication means that after the execution of the mechanism only an identified entity can be in possession of the correct shared secret key.

Key agreement between two entities A and B takes place in a context shared by the two entities. The context consists of sets S_1 and S_2, and a key agreement function F. The function F shall satisfy the following requirements:

a) $\mathrm{F} : S_1 \times S_2 \rightarrow S_2$ maps elements (h, g) $S_1 \times S_2$ to S_2, and we write $y = \mathrm{F}(h, g)$.

b) F satisfies the commutativity condition $\mathrm{F}(h_A, \mathrm{F}(h_B, g)) = \mathrm{F}(h_B, \mathrm{F}(h_A, g))$.

c) It is computationally intractable to find $\mathrm{F}(h_1, \mathrm{F}(h_2, g))$ from $\mathrm{F}(h_1, g)$, $\mathrm{F}(h_2, g)$ and g. This implies that $\mathrm{F}(\cdot, g)$ is a one-way function.

d) The entities A and B share a common element g in S_2 which may be publicly known.

e) The entities acting in this setting can efficiently compute function values $\mathrm{F}(h, g)$ and can efficiently generate random elements in S_1. Depending on the particular key agreement mechanism, further conditions may be imposed.

NOTE 1 Examples for the function F are given in Annex D and Annex E. See also ISO/IEC 15946-1.

NOTE 2 As discussed in Clause 6, in practical implementations of the key agreement mechanisms the shared secret key should be subject to further processing.

NOTE 3 It will in general be necessary to check the received function values $\mathrm{F}(h, g)$ for weak values. If such values are encountered, the protocol shall be aborted.

10.3 Key agreement between three parties

This clause applies to the Key Agreement mechanism 11.12 that describes key agreement between three parties. Key agreement between three parties is the process of establishing a shared secret key among three entities A, B, and C in such a way that none of them can predetermine the value of the shared secret key. Key agreement among three entities A, B, and C takes place in a context shared by the three entities. The context consists of sets S_1, S_2, and S_3, a function F, and a function FP. The functions F and FP shall satisfy the following requirements:

— F : $S_1 \times S_2 \rightarrow S_2$ maps elements (h, g) $S_1 \times S_2$ to S_2, and we write $y = \mathrm{F}(h, g)$.

— F satisfies the commutativity condition $\mathrm{F}(h_A, \mathrm{F}(h_B, g)) = \mathrm{F}(h_B, \mathrm{F}(h_A, g))$.

— It is computationally intractable to find $\mathrm{F}(h_1, \mathrm{F}(h_2, g))$ from $\mathrm{F}(h_1, g)$, $\mathrm{F}(h_2, g)$ and g. This implies that $\mathrm{F}(\cdot, g)$ is a one-way function.

— FP : $S_1 \times S_2 \times S_2 \rightarrow S_3$ maps an element $(h_C, \mathrm{F}(h_A, g), \mathrm{F}(h_B, g))$ $S_1 \times S_2 \times S_2$ to an element of S_3, and we write $z = \mathrm{FP}(h_C, \mathrm{F}(h_A, g), \mathrm{F}(h_B, g))$. ISO/IEC 15946-1 shall be referred for the relation between F and FP.

— FP satisfies the commutativity condition

  — $\mathrm{FP}(h_C, \mathrm{F}(h_A, g), \mathrm{F}(h_B, g)) = \mathrm{FP}(h_C, \mathrm{F}(h_B, g), \mathrm{F}(h_A, g)) = \mathrm{FP}(h_B, \mathrm{F}(h_A, g), \mathrm{F}(h_C, g))$

  $= \mathrm{FP}(h_A, \mathrm{F}(h_B, g), \mathrm{F}(h_C, g)) = \mathrm{FP}(h_A, \mathrm{F}(h_C, g), \mathrm{F}(h_B, g)) = \mathrm{FP}(h_B, \mathrm{F}(h_C, g), \mathrm{F}(h_A, g))$.

— It is computationally intractable to find $\mathrm{FP}(h_C, \mathrm{F}(h_A, g), \mathrm{F}(h_B, g))$ from $\mathrm{F}(h_A, g)$, $\mathrm{F}(h_B, g)$, $\mathrm{F}(h_C, g)$, and g. This implies that $\mathrm{F}(\cdot, p_A, p_B)$ is a one-way function.

— The entities *A*, *B*, and *C* share a common element g in S_2 which may be publicly known.

— The entities acting on this setting can efficiently compute function values $\mathrm{F}(h, g)$ and $\mathrm{FP}(h_C, \mathrm{F}(h_B, g), \mathrm{F}(h_A, g))$, and can efficiently generate random elements in S_1. Depending on the particular key agreement mechanism, further conditions may be imposed.

NOTE 4 An example of a possible function FP is given in Annex F.

NOTE 5 As discussed in Clause 6, in practical implementations of the key agreement mechanisms, the shared secret key should be subject to further processing. A derived shared secret key should be computed by:(1) by extracting bits from the shared secret key K_{ABC} directly, or (2) by passing the shared secret key K_{ABC} and optionally other nonsecret data through a one-way function and extracting bits from the output.

10.4 Secret key transport

Secret key transport (often abbreviated to "key transport") is the process of transferring a secret key, chosen by one entity (or a trusted centre), to another entity, suitably protected by asymmetric cryptographic encryption.

10.5 Public key transport

Public key transport makes an entity's public key available to other entities in an authenticated fashion. Authenticated distribution of public keys is an essential security requirement. This distribution can be achieved in two main ways:

a) Public key distribution without a trusted third party.

b) Public key distribution involving a trusted third party, such as a certification authority.

The public key of an entity *A* is part of the public key information of entity *A*. The public key information includes at least entity *A*'s distinguishing identifier and entity *A*'s public key.

11 Key agreement

11.1 Key agreement mechanism 1

This key agreement mechanism non-interactively establishes a shared secret key between entities A and B with mutual implicit key authentication. The following requirements shall be satisfied:

— Each entity X has a private key agreement key h_X in S_1 and a public key agreement key p_X = F(h_X, g).

— Each entity has access to an authenticated copy of the public key agreement key of the other entity. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 1 is summarised in Figure 1.

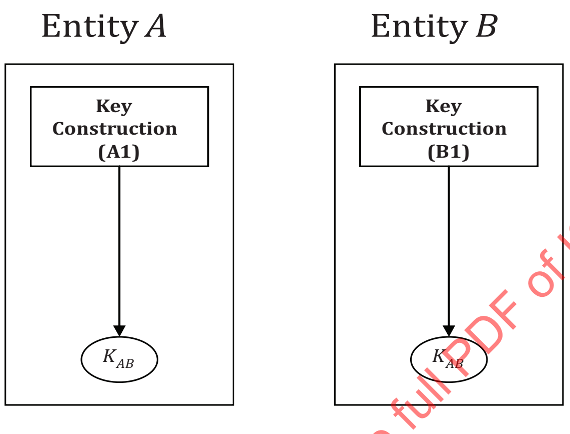

Figure 1 — Key Agreement Mechanism 1

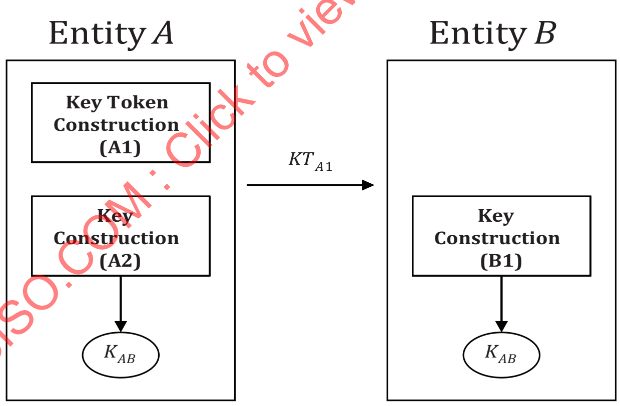

Figure 2 — Key Agreement Mechanisms 2, 8

Key Construction (A1) Entity A computes, using its own private key agreement key h_A and entity B's public key agreement key p_B, the shared secret key as K_{AB} = F(h_A, p_B).

Key Construction (B1) Entity B computes, using its own private key agreement key h_B and entity A's public key agreement key p_A, the shared secret key as K_{AB} = F(h_B, p_A).

As a consequence of requirements on F specified in Clause 10, the two computed values for the key K_{AB} are identical.

NOTE 1 The number of passes is 0.

NOTE 2 This mechanism provides mutual implicit key authentication. However, a zero-pass protocol such as this will always generate the same key. One way to eliminate this problem is to ensure that the key is only used once. Furthermore, the use of a unique initialization vector with each utilization of the key can also solve this problem.

NOTE 3 This mechanism does not provide key confirmation.

NOTE 4 This mechanism is a key agreement mechanism, since the established key is a one-way function of the private key agreement keys h_A and h_B of entities A and B, respectively. However, one entity might learn the other entity's public key prior to choosing their private key. As described in Clause 8, such an entity can select approximately s bits of the established key, at the cost of generating 2^s candidate values for their private key agreement key in the interval between discovering the other entity's public key and choosing their own private key.

NOTE 5 Examples of this mechanism (known as Diffie-Hellman key agreement) are given in Annexes D.2, D.3, and E.3

11.2 Key agreement mechanism 2

This key agreement mechanism establishes a shared secret key in one pass between entities A and B with implicit key authentication from entity B to entity A, but no entity authentication from entity A to entity B (i.e., entity B does not know with whom it has established the shared secret key). The following requirements shall be satisfied:

— Entity B has a private key agreement key h_B in S_1 and a public key agreement key $p_B = F(h_B, g)$.

— Entity A has access to an authenticated copy of entity B's public key agreement key p_B. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 2 is summarised in Figure 2.

Key Token Construction (A1) Entity A randomly and secretly generates r in S_1, computes $F(r, g)$ and sends the key token $KT_{A1} = F(r, g)||\text{Text}$ to entity B

Key Construction (A2) Entity A computes the shared key as $K_{AB} = F(r, p_B)$.

Key Construction (B1) Entity B extracts $F(r,g)$ from the received key token KT_{A1} and computes the shared secret key $K_{AB} = F(h_B, F(r, g))$.

As a consequence of the requirements on F specified in Clause 10, the two computed values for the key K_{AB} are identical.

NOTE 1 The number of passes is 1.

NOTE 2 This mechanism provides implicit key authentication from entity B to entity A (entity B is the only entity other than entity A who can compute the shared secret key).

NOTE 3 This mechanism does not provide key confirmation.

NOTE 4 This mechanism is a key agreement mechanism, since the established key is a one-way function of a random value r supplied by entity A and entity B's private key agreement key. As discussed in Clause 8, since entity A could learn entity B's public key prior to choosing the value r, entity A may select approximately s bits of the established key, at the cost of generating 2^s candidate values for r in the interval between discovering entity B's public key and sending KT_{A1}.

NOTE 5 Examples of this mechanism (known as ElGamal key agreement) are described in Annexes D.4 and E.4.

NOTE 6 As entity B receives the information necessary to compute the key K_{AB} from entity A, which has not been authenticated, use of K_{AB} by entity B should be restricted to functions not requiring trust in entity A's authenticity, such as decryption and generation of message authentication codes.

11.3 Key agreement mechanism 3

This key agreement mechanism establishes a shared secret key in one pass between entities *A* and *B* with mutual implicit key authentication, and entity authentication of entity *A* to entity *B*. The following requirements shall be satisfied:

— Entity *A* has an asymmetric signature system (S_A, V_A).

— Entity *B* has access to an authenticated copy of the public verification transformation V_A. This may be achieved using the mechanisms described in Clause 13.

— Entity *B* has a key agreement scheme with keys (h_B , p_B).

— Entity *A* has access to an authenticated copy of the public key agreement key p_B of entity *B*. This may be achieved using the mechanisms described in Clause 13.

— (Optional) If used, the TVP shall either be a time stamp or a sequence number. If time stamps are used, secure and synchronized time clocks are required; if sequence numbers are used, the ability to maintain and verify bilateral counters is required.

— The entities *A* and *B* have agreed on a MAC function and a way to use K_{AB} as the key for this MAC function. ISO/IEC 9797[5] is referred for a MAC function.

Key agreement mechanism 3 is summarised in Figure 3.

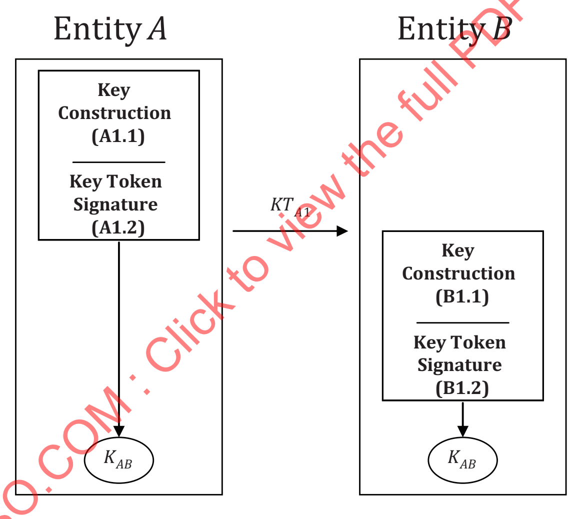

Figure 3 — Key Agreement Mechanism 3

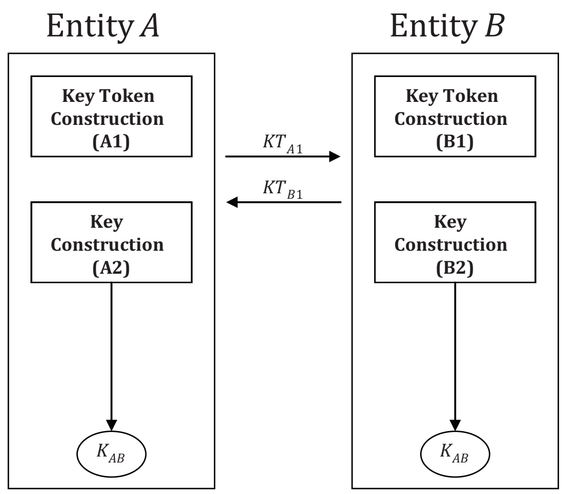

Figure 4 — Key Agreement Mechanisms 4, 5, 9

Key Construction (A1.1) Entity *A* randomly and secretly generates *r* in S_1 and computes F(*r*, *g*). Entity *A* computes the shared secret key as K_{AB} = F(*r*, p_B).

Using the shared secret key K_{AB}, entity *A* computes a MAC on the concatenation of the sender's distinguishing identifier for entity *A* and an optional TVP, a time stamp or a sequence number.

Key Token Signature (A1.2) Entity *A* signs the MAC, using its private signature transformation S_A. Then entity *A* forms the key token, consisting of the sender's distinguishing identifier for entity *A*, the key input F(*r*, *g*), the (optional) TVP, the signed MAC, and some optional data, i.e.

$$KT_{A1} = A||F(r, g)||TVP|| S_A(MAC_{K_{AB}}(A||TVP))||Text1$$

and sends it to entity *B*.

Key Construction (B1.1) Entity *B* extracts F(*r*, *g*) from the received key token and computes the shared secret key, using its private key agreement key h_B, K_{AB} = F(h_B, F(*r*, *g*)).

Using the shared secret key K_{AB}, entity *B* computes the MAC on the sender's distinguishing identifier for entity *A* and the (optional) TVP.

Signature Verification (B1.2) Entity *B* uses the sender's public verification transformation V_A to verify entity *A*'s signature and thus the integrity and origin of the received key token KT_{A1}. Then entity *B* validates the timeliness of the token (by inspection of the (optional) TVP).

NOTE 1 The number of passes is 1.

NOTE 2 This mechanism provides explicit key authentication from entity *A* to entity *B* and implicit key authentication from entity *B* to entity *A*.

NOTE 3 This mechanism provides key confirmation from entity *A* to entity *B*.

NOTE 4 This mechanism is a key agreement mechanism, since the established key is a one-way function of a random value *r* supplied by entity *A* and entity *B*'s private key agreement key. As discussed in Clause 8, since entity *A* could learn entity *B*'s public key prior to choosing the value *r*, entity *A* can select approximately *s* bits of the established key, at the cost of generating 2^s candidate values for *r* in the interval between discovering entity *B*'s public key and sending KT_{A1}.

NOTE 5 The (optional) TVP prevents replay of the key token from entity *A* to entity *B*.

NOTE 6 Examples of this mechanism (known as Nyberg-Rueppel key agreement) are described in Annex D.5 and E.5

NOTE 7 If Text1 is used to transfer entity *A*'s public key certificate, then requirement 2 at the beginning of 11.3 can be relaxed to the requirement that entity *B* is in possession of an authenticated copy of the CA's public verification key.

11.4 Key agreement mechanism 4

This key agreement mechanism establishes a shared secret key in two passes between entities *A* and *B* with joint key control without prior exchange of keying information. This mechanism provides neither entity authentication nor key authentication.

Key agreement mechanism 4 is summarised in Figure 4.

Key Token Construction (A1) Entity *A* randomly and secretly generates r_A in S_1, computes $F(r_A, g)$, constructs the key token $KT_{A1} = F(r_A, g)||\text{Text1}$, and sends it to entity *B*.

Key Token Construction (B1) Entity *B* randomly and secretly generates r_B in S_1, computes $F(r_B, g)$, constructs the key token $KT_{B1} = F(r_B, g)||\text{Text2}$, and sends it to entity *A*.

Key Construction (A2) Entity *A* extracts $F(r_B, g)$ from the received key token KT_{B1} and computes the shared secret key $K_{AB} = F(r_A, F(r_B, g))$.

Key Construction (B2) Entity *B* extracts $F(r_A, g)$ from the received key token KT_{A1} and computes the shared secret key $K_{AB} = F(r_B, F(r_A, g))$.

NOTE 1 The number of passes is 2.

NOTE 2 This mechanism does not provide implicit or explicit key authentication. However, this mechanism can be useful in environments where authenticity of the key tokens is verified using other means. For instance, a hash-code of the key tokens could be exchanged between the entities using a second communication channel. See also Public Key Transport Mechanism 2. Another example of entity authentication is using mechanisms specified in.[6]

NOTE 3 A separate channel or means shall exist whereby the key tokens can be verified.

NOTE 4 This mechanism provides no key confirmation.

NOTE 5 This mechanism is a key agreement mechanism, since the established key is a one-way function of random values r_A and r_B supplied by entities *A* and *B* respectively. As discussed in Clause 8, since entity *B* could learn $F(r_A, g)$ prior to choosing the value r_B, entity *B* can select approximately *s* bits of the established key, at the cost of generating 2^s candidate values for r_B in the interval between receiving KT_{A1} and sending KT_{B1}.

NOTE 6 Examples of this mechanism (known as Diffie-Hellman key agreement) are described in Annexes D.6 and E.7.

11.5 Key agreement mechanism 5

This key agreement mechanism establishes a shared secret key in two passes between entities *A* and *B* with mutual implicit key authentication and joint key control. The following requirements shall be satisfied:

— Each entity *X* has a private key agreement key h_X in S_1 and a public key agreement key $p_X = F(h_X, g)$.

— Each entity has access to an authenticated copy of the public key agreement key of the other entity. This may be achieved using the mechanisms described in Clause 13.

— Both entities have agreed on a common one-way function w.

Key agreement mechanism 5 is summarised in Figure 4.

Key Token Construction (A1) Entity *A* randomly and secretly generates r_A in S_1, computes $F(r_A, g)$ and sends the key token $KT_{A1} = F(r_A, g)||\text{Text1}$ to entity *B*.

Key Token Construction (B1) Entity B randomly and secretly generates r_B in S_1, computes $F(r_B, g)$ and sends the key token $KT_{B1} = F(r_B, g)||Text2$ to entity A.

Key Construction (B2) Entity B extracts $F(r_A, g)$ from the received key token KT_{A1} and computes the shared secret key as $K_{AB} = w(F(h_B, F(r_A, g))||F(r_B, p_A))$ where w is a one-way function.

Key Construction (A2) Entity A extracts $F(r_B, g)$ from the received key token KT_{B1} and computes the shared secret key as $K_{AB} = w(F(r_A, p_B)||F(h_A, F(r_B, g)))$.

NOTE 1 The number of passes is 2.

NOTE 2 This mechanism provides mutual implicit key authentication. If the data field Text2 contains a MAC (on known data) computed using the key K_{AB}, then this mechanism provides explicit key authentication from entity B to entity A.

NOTE 3 If the data field Text2 contains a MAC (on known data) computed using the key K_{AB}, then this mechanism provides key confirmation from entity B to entity A.

NOTE 4 This mechanism is a key agreement mechanism, since the established key is a one-way function of random values r_A and r_B supplied by entities A and B respectively.

NOTE 5 Examples of this key agreement mechanism (known as the Matsumoto-Takashima-Imai A(0) key agreement scheme) are described in Annexes D.7 and E.6. Another example is known as the Goss protocol.

NOTE 6 If Text1 and Text2 contain the public key certificates of entity A's and B's key agreement keys, respectively, then the requirement 2 at the beginning of 11.5 can be replaced by the requirement that each entity is in possession of an authenticated copy of the CA's public verification key.

NOTE 7 Under certain circumstances this mechanism may be subject to a source substitution attack.[30] If this is a concern, this type of attack can be avoided by ensuring that as part of the process of submitting a public key to a CA for certification, the submitter proves possession of the corresponding private key. This type of attack is slightly more serious in the case of protocols based on elliptic curves.[26]

11.6 Key agreement mechanism 6

This key agreement mechanism establishes a shared secret key in two passes between entities A and B with mutual implicit key authentication and joint key control. It is based on the use of both an asymmetric encryption scheme and a signature system. The following requirements shall be satisfied:

a) Entity A has an asymmetric encryption system with transformations (E_A, D_A).

b) Entity B has an asymmetric signature system with transformations (S_B, V_B).

c) Entity A has access to an authenticated copy of entity B's public verification transformation V_B. This may be achieved using the mechanisms described in Clause 13.

d) Entity B has access to an authenticated copy of entity A's public encryption transformation E_A. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 6 is summarised in Figure 5.

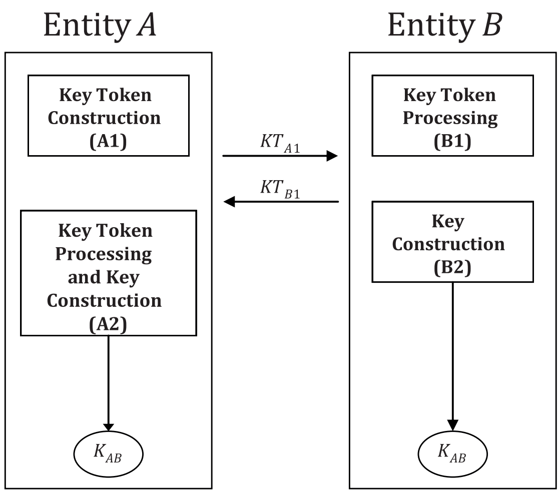

Figure 5 — Key Agreement Mechanism 6

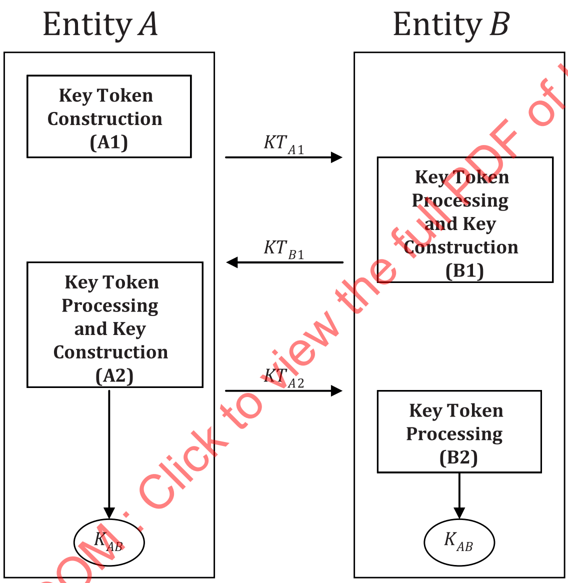

Figure 6 — Key Agreement Mechanism 7

Key Token Construction (A1) Entity *A* generates a random number r_A, constructs the key token KT_{A1} = r_A||Text1, and sends it to entity *B*.

Key Token Processing (B1) Entity *B* generates a random number r_B and signs a data block consisting of the distinguishing identifier for entity *A*, the random number r_A, the random number r_B and some optional data Text2 using its private signature transformation S_B, to obtain $BS = S_B(A||r_A||r_B||\text{Text2})$.

Entity *B* then encrypts a data block consisting of its distinguishing identifier (optional), the signed block *BS*, and some optional data Text3 using entity *A*'s public encryption transformation E_A. Entity *B* then sends the key token $KT_{B1} = E_A(BS||\text{Text3})||\text{Text4}$ back to entity *A*, or entity *B* may include the identifier for *B* as $KT_{B1}= E_A(B||BS||\text{Text3})$ ||Text4.

Key Construction (B2) The shared secret key consists of all or part of entity *B*'s signature ∑ contained in the signed block *BS* (see Note 2 in Clause 4), after passing through a key derivation function.

Key Token Processing and Key Construction (A2) Entity *A* decrypts the key token KT_{B1} using its private decryption transformation D_A, optionally checks the sender identifier, and uses entity *B*'s public verification transformation V_B to verify the digital signature of the signed block *BS*. Then entity *A* checks the recipient identifier and that the random number r_A in the signed block *BS* equals the random number r_A sent in token KT_{A1}. If all checks are successful, entity A accepts all or part of entity B's signature of the signed block BS used with a key derivation function as the shared secret key.

NOTE 1 The number of passes is 2.

NOTE 2 The part of the signature Σ that is to be used as the basis of the secret key established between entities A and B shall be agreed in advance.

NOTE 3 This mechanism provides implicit key authentication from entity A to entity B and explicit key authentication from entity B to entity A.

NOTE 4 If the data field Text3 contains a MAC (on known data) computed using the key K_{AB}, then this mechanism provides key confirmation from entity B to entity A.

NOTE 5 This mechanism is a key agreement mechanism, since the established key is a one-way function of random values r_A and r_B supplied by entities A and B respectively. As discussed in Clause 8, since entity B could learn $F(r_A, g)$ prior to choosing the value r_B, entity B can select approximately s bits of the established key, at the cost of generating 2^s candidate values for r_B in the interval between receiving KT_{A1} and sending KT_{B1}.

NOTE 6 This mechanism is derived from Beller and Yacobi's two pass protocol described in Annex D.8.

NOTE 7 If Text1 and Text4 contain a public key certificate for entity A's encryption key and a public key certificate for entity B's verification key, respectively, then the requirements 3 and 4 at the beginning of 11.6 can be relaxed to the requirement that each entity is in possession of an authenticated copy of the CA's public verification key.

NOTE 8 A significant feature of this scheme is that the identity of entity B can remain anonymous to eavesdroppers, a property of potential significance in a wireless communication environment.

11.7 Key agreement mechanism 7

This key agreement mechanism is based on the three-pass authentication mechanism of ISO/IEC 9798-3[6] and establishes a shared secret key between entities A and B with mutual authentication. The following requirements shall be satisfied:

— Each entity X has an asymmetric signature system (S_X, V_X).

— Both entities have access to an authenticated copy of the public verification transformation of the other entity. This may be achieved using the mechanisms described in Clause 13.

— The two entities have agreed on a common MAC function.

Key agreement mechanism 7 is summarised in Figure 6.

Key Token Construction (A1) Entity A randomly and secretly generates r_A in S_1, computes $F(r_A, g)$, constructs the key token $KT_{A1} = F(r_A, g)||\text{Text1}$, and sends it to entity B.

Key Token Processing and Key Construction (B1) Entity B randomly and secretly generates r_B in S_1, computes $F(r_B, g)$, computes the shared secret key as $K_{AB} = F(r_B, F(r_A, g))$, and constructs the signed key token

$$KT_{B1} = S_B(DB_1)||\text{MAC}_{K_{AB}}(DB_1)||\text{Text3},$$

where $DB_1 = F(r_B, g)||F(r_A, g)||A||\text{Text2}$, and sends it back to entity A.

Key confirmation is provided by including $\text{MAC}_{K_{AB}}(DB_1)$ in KT_{B1}. Alternatively, if both parties have a common symmetric encryption system, key confirmation can be obtained by replacing KT_{B1} with $KT_{B1} = F(r_B, g)||E_{K_{AB}}(S_B(DB_1))$, where E is a suitable symmetric encryption function.

Key Token Processing and Key Construction (A2) Entity A verifies entity B's signature on the key token KT_{B1} using entity B's public verification key, and then verifies entity A's distinguishing identifier and the value $F(r_A, g)$ sent in step (A1). If the checks are successful, entity A proceeds to compute the shared secret key as $K_{AB} = F(r_A, F(r_B, g))$.

Using K_{AB}, entity A verifies $\text{MAC}_{K_{AB}}(DB_1)$. Then entity A constructs the signed key token

$$KT_{A2} = \text{S}_A(DB_2)||\text{MAC}_{K_{AB}}(DB_2)||\text{Text5},$$

where $DB_2 = \text{F}(r_A, g)||\text{F}(r_B, g)||\text{B}||\text{Text4}$, and sends it to entity B.

Key confirmation is provided by including $\text{MAC}_{K_{AB}}(DB_2)$ in KT_{A2}. Alternatively, key confirmation can be obtained by replacing KT_{A2} with $KT_{A2} = E_{K_{AB}}(\text{S}_A(DB_2))$.

Key Token Processing (B2) Entity B verifies entity A's signature on the key token KT_{A2}, using entity A's public verification key, then verifies entity B's distinguishing identifier and that the values $\text{F}(r_A, g)$ and $\text{F}(r_B, g)$ agree with the values exchanged in the previous steps. If the checks are successful, entity B verifies $\text{MAC}_{K_{AB}}(DB_2)$ using $K_{AB} = \text{F}(r_B, \text{F}(r_A, g))$.

NOTE 1 The number of passes is 3.

NOTE 2 This mechanism provides mutual explicit key authentication and mutual entity authentication.

NOTE 3 This mechanism provides mutual key confirmation.

NOTE 4 This mechanism is a key agreement mechanism, since the established key is a one-way function of random values r_A and r_B supplied by entities A and B respectively. As discussed in Clause 8, since entity B could learn $\text{F}(r_A, g)$ prior to choosing the value r_B, entity B can select approximately s bits of the established key, at the cost of generating 2^s candidate values for r_B in the interval between receiving KT_{A1} and sending KT_{B1}.

NOTE 5 Examples of this mechanism (known as the Diffie-Hellman scheme) can be constructed by combining the examples in Annex E.9 with use of a digital signature scheme, such as one of those specified in ISO/IEC 9796 and ISO/IEC 14888.

NOTE 6 This mechanism conforms to ISO/IEC 9798-3.[6] KT_{A1}, KT_{B1}, and KT_{A2} are identical to the tokens sent in the three pass authentication mechanism. The TVPs are also identical, with the following changes of use: the TVP R_A is set to the value $\text{F}(r_A, g)$; and the TVP R_B is set to the value $\text{F}(r_B, g)$.

NOTE 7 If the data fields Text1 and Text3 (or Text5 and Text3) contain the public key certificates of entities A and B, respectively, then the second requirement at the beginning of 11.7 can be relaxed to the requirement that all entities are in possession of an authenticated copy of the CA's public verification key.

NOTE 8 If a signature mechanism with text hashing is used, then $\text{F}(r_A, g)$ and/or $\text{F}(r_B, g)$ need not be sent in key token KT_{B1}. Similarly, neither $\text{F}(r_A, g)$ nor $\text{F}(r_B, g)$ need be sent in key token KT_{A2}. However, care shall be taken that the random numbers are included in the computation of the respective signatures.

NOTE 9 Key confirmation can alternatively be achieved by encrypting part of the signature. In this case, the third requirement at the beginning of 11.7 does not apply.

11.8 Key agreement mechanism 8

This key agreement mechanism uses elliptic curve cryptography, and establishes a shared secret key in one pass between entities A and B with mutual implicit key authentication. The following requirements shall be satisfied:

— Each entity X has a private key agreement key h_X in S_1 and a public key agreement key $P_X = \text{F}(h_X, G)$.

— Each entity has access to an authenticated copy of the public key agreement key of the other entity. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 8 is summarised in Figure 2.

The values l and j are used for cofactor multiplication as explained in Clause 7. A function is also required to convert an elliptic point P to an integer. An example of such a function is $\pi(P) = (\text{X}(P) \bmod 2^{\lceil \rho/2 \rceil}) + 2^{\lceil \rho/2 \rceil}$, where $\rho = \lceil \log_2 n \rceil$ and $\text{X}(P)$ is the x-coordinate of the point P.

Key token construction (A1) Entity A randomly and secretly generates r_A in S_1, computes $\text{F}(r_A, G)$, constructs the key token $KT_{A1} = \text{F}(r_A, G)$, and sends it to entity B.

Key construction (A2) Entity A computes the shared key as

$$K_{AB} = ((r_A + \pi(KT_{A1})h_A)\cdot l)(j\cdot(P_B + \pi(P_B)P_B)).$$

Key construction (B1) Entity B computes the shared key as

$$K_{AB} = ((h_B + \pi(P_B)h_B)\cdot l)(j\cdot(KT_{A1} + \pi(KT_{A1})P_A)).$$

NOTE 1 The number of passes is 1.

NOTE 2 This mechanism provides mutual implicit key authentication.

NOTE 3 An example of this mechanism (known as MQV key agreement) is described in Annex E.11.

11.9 Key agreement mechanism 9

This key agreement mechanism uses elliptic curve cryptography and establishes a shared secret key in two passes between entities A and B with mutual implicit key authentication. The following requirements shall be satisfied:

a) Each entity X has a private key agreement key h_X in S_1 and a public key agreement key $P_X = F(h_X, G)$.

b) Each entity has access to an authenticated copy of the public key agreement key of the other entity. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 9 is summarised in Figure 4.

The values l and j are used for cofactor multiplication as explained in Clause 7. A function is also required to convert an elliptic point P to an integer. An example of such a function is $\pi(P) = (X(P) \bmod 2^{\lceil \rho/2 \rceil}) + 2^{\lceil \rho/2 \rceil}$, where $\rho = \lceil \log_2 n \rceil$ and $X(P)$ is the x-coordinate of the point P.

Key token construction (A1) Entity A randomly and secretly generates r_A in S_1, computes $F(r_A, G)$, constructs the key token $KT_{A1} = F(r_A, G)$, and sends it to entity B.

Key token construction (B1) Entity B randomly and secretly generates r_B in S_1, computes $F(r_B, G)$, constructs the key token $KT_{B1} = F(r_B, G)$, and sends it to entity A.

Key construction (A2) Entity A computes the shared secret key as

$$K_{AB} = ((r_A + \pi(KT_{A1})h_A)\cdot l)(j\cdot(KT_{B1} + \pi(KT_{B1})P_B)).$$

Key construction (B2) Entity B computes the shared secret key as

$$K_{AB} = ((r_B + \pi(KT_{B1})h_B)\cdot l)(j\cdot(KT_{A1} + \pi(KT_{A1})P_A)).$$

NOTE 1 The number of passes is 2.

NOTE 2 This mechanism provides mutual implicit key authentication.

NOTE 3 An example of this mechanism (known as MQV key agreement with two passes) is described in Annex E.12.

NOTE 4 Under certain circumstances this mechanism may be subject to a source substitution attack.[26] If this is a concern, such an attack can be avoided by adding delay detection. Other countermeasures are described in [26].

11.10 Key agreement mechanism 10

This key agreement mechanism uses elliptic curve cryptography and establishes a shared secret key in three passes between entities *A* and *B* with mutual implicit key authentication. The following requirements shall be satisfied:

— Each entity *X* has a private key agreement key h_X in S_1 and a public key agreement key $P_X = F(h_X, G)$.

— Each entity has access to an authenticated copy of the public key agreement key of the other entity. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 10 is summarised in Figure 7.

The values *l* and *j* are used for cofactor multiplication as explained in Clause 7. A function is also required to convert an elliptic point *P* to an integer. An example of such a function is $\pi(P) = (X(P) \bmod 2^{\lceil \rho/2 \rceil}) + 2^{\lceil \rho/2 \rceil}$, where $\rho = \lceil \log_2 n \rceil$ and X(*P*) is the *x*-coordinate of the point *P*.

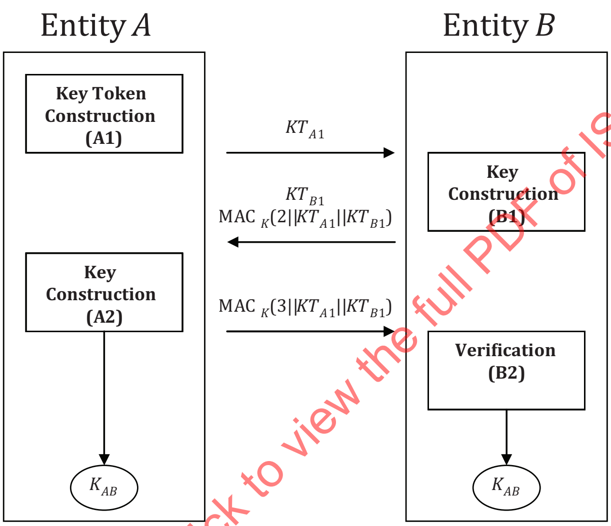

Figure 7 — Key Agreement Mechanism 10

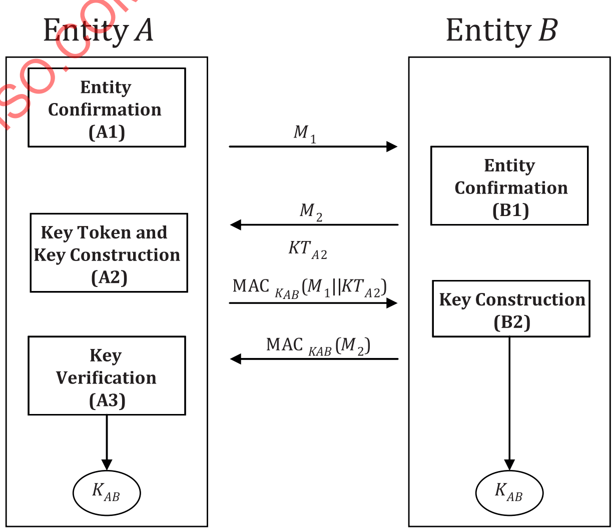

Figure 8 — Key Agreement Mechanism 11

Key token construction (A1) Entity *A* randomly and secretly generates r_A in S_1, computes $F(r_A, G)$, constructs the key token $KT_{A1} = F(r_A, G)$, and sends it to entity *B*.

Key construction (B1) Entity B randomly and secretly generates r_B in S_1, computes $F(r_B, G)$, and constructs the key token $KT_{B1} = F(r_B, G)$.

Entity B computes the shared secret key as

$$K_{AB} = ((r_B + \pi(KT_{B1})h_B)\cdot l)(j\cdot(KT_{A1} + \pi(KT_{A1})P_A)).$$

Entity B then computes the key $K = \mathrm{kdf}(K_{AB})$. Entity B further constructs $\mathrm{MAC}_K(2||KT_{A1}||KT_{B1})$, and sends KT_{B1} and $\mathrm{MAC}_K(2 || KT_{A1} || KT_{B1})$ to entity A.

Key construction (A2) Entity A computes the shared secret key as

$$K_{AB} = ((r_A + \pi(KT_{A1})h_A)\cdot l)(j\cdot(KT_{B1} + \pi(KT_{B1})P_B)).$$

Entity A computes the key $K = \mathrm{kdf}(K_{AB})$. Entity A computes $\mathrm{MAC}_K(2||KT_{A1}||KT_{B1})$ and verifies what was sent by entity B. Entity A then computes $\mathrm{MAC}_K(3||KT_{A1}||KT_{B1})$, and sends it to entity B.

Verification (B2) Entity B computes $\mathrm{MAC}_K(3||KT_{A1}||KT_{B1})$.

NOTE 1 The number of passes is 3.

NOTE 2 This mechanism provides mutual explicit key authentication.

NOTE 3 An example of this mechanism (known as MQV key agreement with three passes) is described in Annex E.13.

11.11 Key agreement mechanism 11

This key agreement mechanism establishes a shared key in four passes between entities A and B. The following requirements shall be satisfied:

— Entity B has an asymmetric encryption system with transformation (E_B, D_B).

— Entity A has access to an authenticated copy of the public verification transformation necessary to verify $Cert_B$.

— Both entities have agreed on a common key derivation function kdf.

Key agreement mechanism 11 is summarised in Figure 8.

Entity Confirmation (A1): Entity A chooses a random integer r_A, and sends a message $M_1 = (r_A||\mathrm{Text1})$ to entity B.

Entity Confirmation (B1): Entity B chooses a random integer r_B, and sends $M_2 = (r_B||Cert_B||\mathrm{Text2})$ to entity A.

Key Token and Key Construction (A2): Entity A verifies $Cert_B$ to obtain a trusted copy of entity B's public key. Entity A then generates a random integer r'_A and computes the shared key $K_{AB} = \mathrm{kdf}(r_A, r_B, r'_A)$.

Entity A then sends the key token $KT_{A2} = E_B(r'_A)$ and $\mathrm{MAC}_{K_{AB}}(M_1||KT_{A2})$ to entity B.

Key Construction (B2): Entity B decrypts KT_{A2} and computes the shared key $K_{AB} = \mathrm{kdf}(r_A, r_B, r'_A)$.

Entity B computes $\mathrm{MAC}_{K_{AB}}(M_1 || KT_{A2})$ and compares it with the received MAC value. Entity B sends $\mathrm{MAC}_{K_{AB}}(M_2)$ to entity A.

Key Verification (A3): Entity A computes $\mathrm{MAC}_{K_{AB}}(M_2)$ and compares it with the received MAC value.

NOTE 1 The number of passes is 4.

NOTE 2 This mechanism provides B's implicit key authentication to A.

NOTE 3 This mechanism is derived from the Transport Layer Security (TLS) protocol,[15] which can be regarded as an example of this mechanism. In TLS, the key agreement process is known as the TLS handshake phase. In TLS, each entity has a 'cipher suite', i.e. a list of algorithms that the entity supports. Text1 and Text2 are used to exchange these cipher suites as part of a process known as 'cipher suite negotiation'.

11.12 Key agreement mechanism 12

This key agreement mechanism non-interactively establishes a shared secret key among entities *A*, *B*, and *C* with mutual implicit key authentication. The following requirements shall be satisfied:

— Each entity *X* has a private key-agreement key h_X in S_1 and a public key-agreement key $p_X = F(h_X, g)$.

— Each entity has access to an authenticated copy of the public key-agreement key of the other entities. This may be achieved using the mechanisms described in Clause 13.

Key agreement mechanism 12 is summarised in Figure 9.

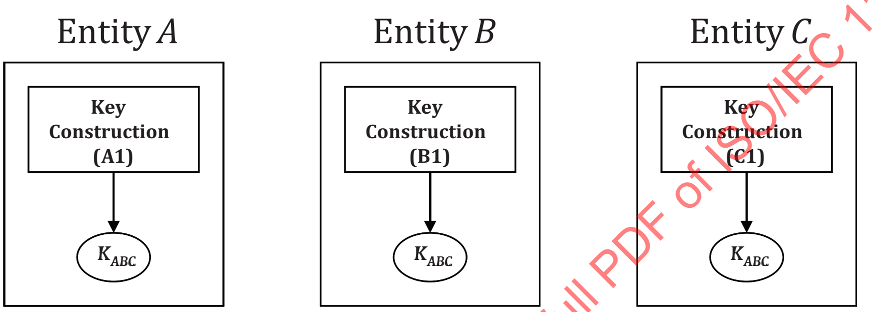

Figure 9 — Key Agreement Mechanism 12

Key Construction (A1) Entity *A* computes, using its own private key-agreement key h_A, entity *B*'s public key-agreement key p_B, and entity *C*'s public key-agreement key p_C, the shared secret key as $K_{ABC} = FP(h_A, p_B, p_C)$.

Key Construction (B1) Entity *B* computes, using its own private key-agreement key h_B, entity *A*'s public key-agreement key p_A, and entity *C*'s public key-agreement key p_C, the shared secret key as $K_{ABC} = FP(h_B, p_C, p_A)$.

Key Construction (C1) Entity *C* computes, using its own private key-agreement key h_C, entity *A*'s public key-agreement key p_A, and entity *B*'s public key-agreement key p_B, the shared secret key as $K_{ABC} = FP(h_C, p_A, p_B)$.

As a consequence of the requirements on functions F and FP specified in Clause 10, the three computed values for the key K_{ABC} are identical.

NOTE 1 The number of passes is 0.

NOTE 2 This mechanism provides mutual implicit key authentication. However, a zero-pass protocol such as this will always generate the same key. One way to eliminate this problem is to ensure that the key is only used once. Furthermore, the use of a unique initialization vector with each utilization of the key can also solve this problem.

NOTE 3 This mechanism does not provide key confirmation.

NOTE 4 This is a key agreement mechanism, since the established key is a one-way function of the private key agreement keys h_A, h_B, and h_C of entities *A*, *B*, and *C* respectively.

NOTE 5 An example of this mechanism (known as Joux key agreement) is given in Annex F.2.

12 Secret key transport

12.1 Secret key transport mechanism 1

This secret key transport mechanism transfers a secret key in one pass from entity *A* to entity *B* with implicit key authentication from entity *B* to entity *A*. The following requirements shall be satisfied:

— Entity *B* has an asymmetric encryption system (E_B,D_B).

— Entity *A* has access to an authenticated copy of entity *B*'s public encryption transformation E_B. This may be achieved using the mechanisms described in Clause 13.

— The optional TVP shall either be a time stamp or sequence number. If time stamps are used, then the entities *A* and *B* need to maintain synchronous clocks. If sequence numbers are used, then entities *A* and *B* shall maintain bilateral counters.

Secret key transport mechanism 1 is summarised in Figure 10.

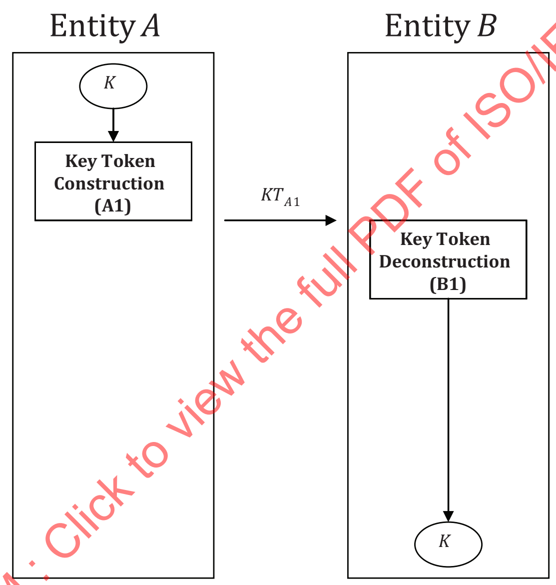

Figure 10 — Secret Key Transport Mechanism 1

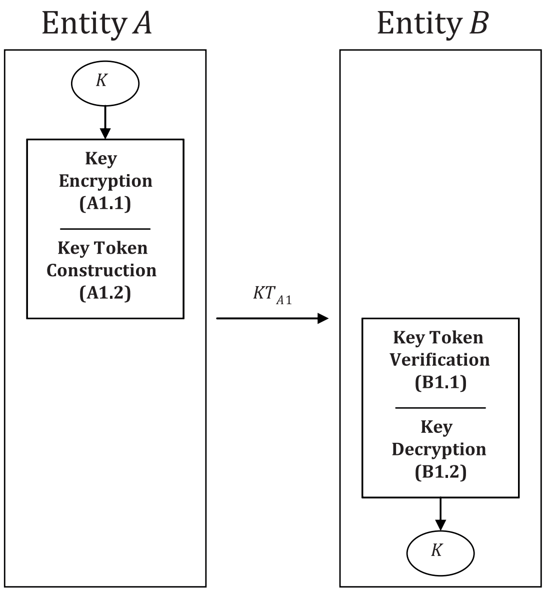

Figure 11 — Secret Key Transport Mechanism 2

Key Token Construction (A1) Suppose *K* is a secret key that entity *A* wishes to securely transfer to entity *B*. Entity *A* constructs a key data block consisting of its distinguishing identifier (optional), the key *K*, an optional TVP and an optional data field Text1. Entity *A* then encrypts the key data block using the receiver's public encryption transformation E_B and sends the key token

$$KT_{A1} = E_B(A||K||TVP||Text1)||Text2$$

to entity *B*.

Key Token Deconstruction (B1) Entity *B* decrypts the encrypted part of the received key token KT_{A1} using its private decryption transformation D_B, recovers the key *K*, checks the optional TVP, and associates the recovered key *K* with the claimed originator entity *A*.

NOTE 1 The number of passes is 1.

NOTE 2 This mechanism provides implicit key authentication from entity *B* to entity *A*, since only entity *B* can possibly recover the key *K*.

NOTE 3 This mechanism does not provide key confirmation.

NOTE 4 Entity *A* can choose the key.

NOTE 5 As entity *B* receives the key *K* from a non-authenticated entity *A*, secure use of *K* by entity *B* should be restricted to functions not requiring trust in entity *A*'s authenticity. For example, decryption and generation of message authentication codes can be performed, whereas encryption and verification of message authentication codes should not.

NOTE 6 An example of this mechanism (known as ElGamal key transfer) is described in Annex G.1. A second example of this mechanism using RSA is described in Annex G.3, and a third example based on Sakai-Kasahara Key Establishment is described in Annex G.6.

12.2 Secret key transport mechanism 2

This secret key transport mechanism is an extension of the one-pass entity authentication mechanism in ISO/IEC 9798-3.[6] It transfers a secret key, encrypted and signed, from entity *A* to entity *B* with explicit key authentication from entity *A* to entity *B* and implicit key authentication from entity *B* to entity *A*. The following requirements shall be satisfied:

— Entity *A* has an asymmetric signature system (S_A, V_A).

— Entity B has an asymmetric encryption system (E_B, D_B).

— Entity A has access to an authenticated copy of entity B's public encryption transformation E_B. This may be achieved using the mechanisms described in Clause 13.

— Entity B has access to an authenticated copy of entity A's public verification transformation V_A. This may be achieved using the mechanisms described in Clause 13.

— The optional TVP shall be either a time stamp or sequence number. If time stamps are used, then the entities A and B need to maintain synchronous clocks or use a Trusted Third Party Time Stamp Authority. If sequence numbers are used then entities A and B shall maintain bilateral counters.

Secret key transport mechanism 2 is summarised in Figure 11.

Key Encryption (A1.1) Suppose K is a secret key that entity A wishes to securely transfer to entity B. Entity A forms the key data block, consisting of the sender's distinguishing identifier, the key K and an optional data field Text1. Entity A then encrypts the key data block with entity B's public encryption transformation E_B and forms the encrypted block $BE = E_B(A||K||\text{Text1})$.

Key Token Construction (A1.2) Entity A forms the token data block, consisting of the recipient's distinguishing identifier, an optional TVP (time stamp or sequence number), the encrypted block BE and the optional data field Text2. Then entity A signs the token data block using its private signature transformation S_A, appends optional Text3, and sends the resulting key token

$$KT_{A1} = S_A(B||TVP||BE||\text{Text2})||\text{Text3}$$

to entity B.

Key Token Verification (B1.1) Entity B uses the sender's public verification transformation V_A to verify the digital signature in the received key token KT_{A1}. Entity B then checks its identifier in KT_{A1} and, optionally, the TVP.

Key Decryption (B1.2) Entity B decrypts the block BE with its private decryption transformation D_B. Entity B then compares the identifier for entity A contained in block BE with the identity of the signing entity. If all checks are successful, entity B accepts the key K.

NOTE 1 The number of passes is 1.

NOTE 2 This mechanism provides entity authentication of entity A to entity B, and implicit key authentication from entity B to entity A.

NOTE 3 This mechanism provides key confirmation from entity A to entity B. Entity B can be sure that it shares the correct key with entity A, but entity A can only be sure that entity B has indeed received the key after it has obtained a positive reply from entity B encrypted using key K.

NOTE 4 The optional TVP provides entity authentication of entity A to entity B and prevents replay of the key token. In order to prevent replay of the key data block BE, an additional TVP can also be included in Text1.

NOTE 5 Entity A can choose the key K_A, since it is the originating entity. Similarly, entity B can choose the key K_B. Joint key control can be achieved by requiring entities A and B to combine two keys K_A and K_B, transported using two instances of the mechanism, to form a shared secret key K_{AB}. An extra pass is required for joint key control. The combination function shall be one-way, otherwise entity A can choose the shared secret key. This mechanism can then be classified as a key agreement mechanism.

NOTE 6 Entity A's distinguishing identifier is included in the encrypted block BE to prevent entity A from misappropriating an encrypted key block intended for use by another entity. Prevention of the attack is achieved by requiring entity B to compare entity A's identifier with entity A's signature on the token.

NOTE 7 In conformance with ISO/IEC 9798-3,[6] entity authentication using a public key algorithm KT_{A1} is compatible with the token sent in the one-pass authentication mechanism. The token accommodates the transfer of the key K through use of the optional text field: Text1 in the ISO/IEC 9798-3[6] mechanism has been replaced by BE || Text2.

NOTE 8 The data field Text3 can be used to deliver the public key certificate of entity *A*. If this is the case, then the fourth requirement at the beginning of 12.2 can be relaxed to the requirement that entity *B* is in possession of an authenticated copy of the CA's public verification key.

NOTE 9 Examples of this mechanism are described in Annexes G.2 and G.5.

12.3 Secret key transport mechanism 3

This secret key transport mechanism transfers a secret key, signed, and encrypted in one pass from entity *A* to entity *B* with unilateral key confirmation. The following requirements shall be satisfied:

a) Entity *A* has an asymmetric signature system (S_A, V_A).

b) Entity *B* has an asymmetric encryption system (E_B, D_B).

c) Entity *A* has access to an authenticated copy of entity B's public encryption transformation E_B. This may be achieved using the mechanisms described in Clause 13.

d) Entity *B* has access to an authenticated copy of entity *A*'s public verification transformation V_A. This may be achieved using the mechanisms described in Clause 13.

e) The optional TVP shall be either a time stamp or a sequence number. If time stamps are used then the entities *A* and *B* need to maintain synchronous clocks. If sequence numbers are used then entities *A* and *B* shall maintain bilateral counters.

Secret key transport mechanism 3 is summarised in Figure 12.

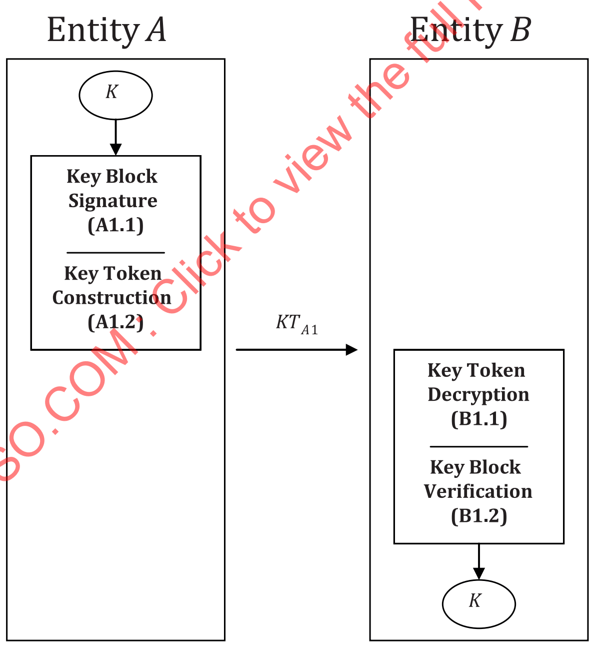

Figure 12 — Secret Key Transport Mechanism 3

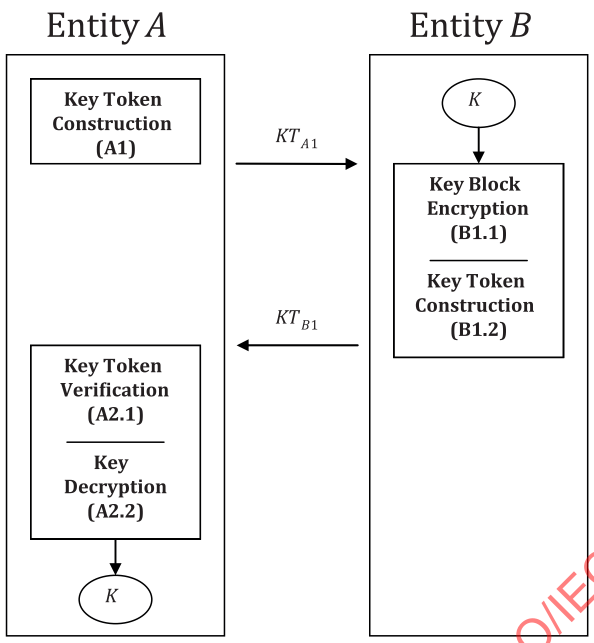

Figure 13 — Secret Key Transport Mechanism 4

Key Block Signature (A1.1) Suppose *K* is a secret key that entity *A* wishes to securely transfer to entity *B*. Entity *A* forms a key data block consisting of the recipient's distinguishing identifier, the key *K*, an optional TVP (sequence number or time stamp), and optional data. Entity *A* then signs the key block using its private signature transformation S_A to generate the signed block $BS = S_A(B||K||TVP||Text1)$.

Key Token Construction (A1.2) Entity *A* forms the token data block, consisting of the signed block *BS* and optional Text2. Then entity *A* encrypts the token data block using the receiver's public encryption transformation E_B, appends optional Text3, and sends the resulting key token

$$KT_{A1} = E_B(BS||Text2)||Text3$$

to entity *B*.

Key Token Decryption (B1.1) Entity *B* decrypts the encrypted part of the received key token KT_{A1} using its private decryption transformation D_B.

Key Block Verification (B1.2) Entity *B* uses the sender's public verification transformation V_A to verify the integrity and origin of *BS*. Entity *B* validates that it is the intended recipient of the token (by inspection of the identifier in *BS*) and, optionally, that the TVP is within acceptable bounds (to verify the token's timeliness). If all verifications are successful, entity *B* accepts the key *K*.

NOTE 1 The number of protocol passes is 1.

NOTE 2 This mechanism provides entity authentication of entity *A* to entity *B*, and implicit key authentication from entity *B* to entity *A*.

NOTE 3 This mechanism provides key confirmation from entity *A* to entity *B*. Entity *B* can be sure that it shares the correct key *K* with entity *A*, but entity *A* can only be sure that entity *B* has indeed received the key after it has obtained a positive reply from entity *B* encrypted using key *K*.

NOTE 4 Entity *A* can choose the key.

NOTE 5 Entity B's distinguishing identifier is included in the signed key block *BS* to explicitly indicate the recipient of the key, thereby preventing misuse of the signed block *BS* by entity *B*.

NOTE 6 The data field Text3 can be used to deliver the public key certificate of entity *A*. If this is the case, then the fourth requirement at the beginning of 12.3 can be relaxed to the requirement that entity *B* is in possession of an authenticated copy of the CA's public verification key.

NOTE 7 If two executions of this secret key transport mechanism are combined (from entity *A* to entity *B* and from entity *B* to entity *A*) then mutual entity authentication and joint key control can be provided (depending on use of the optional TVP).

12.4 Secret key transport mechanism 4

This secret key transport mechanism is based on the two-pass authentication mechanism of ISO/IEC 9798-3,[6] and transfers a key from entity *B* to entity *A*. The following requirements shall be satisfied:

— Entity *A* has an asymmetric encryption system (E_A, D_A).

— Entity *B* has an asymmetric signature system (S_B, V_B).

— Entity *A* has access to an authenticated copy of entity B's public verification transformation V_B. This may be achieved using the mechanisms described in Clause 13.

— Entity *B* has access to an authenticated copy of entity *A*'s public encryption transformation E_A. This may be achieved using the mechanisms described in Clause 13.

Secret key transport mechanism 4 is summarised in Figure 13.

Key Token Construction (A1) Entity *A* generates a random number r_A, constructs the key token KT_{A1} consisting of r_A and an optional data field Text1, $KT_{A1} = r_A||\text{Text1}$ and sends it to entity *B*.

Key Block Encryption (B1.1) Suppose *K* is a secret key that entity *B* wishes to securely transfer to entity *A*. Entity *B* forms a key data block, consisting of the sender's distinguishing identifier, the key *K* and an optional data field Text2. Entity *B* then encrypts the key data block with entity *A*'s public encryption transformation E_A, and forms the encrypted block $BE = E_A(B||K||\text{Text2})$.

Key Token Construction (B1.2) Entity *B* optionally generates a random number r_B and forms the token data block, consisting of the recipient's distinguishing identifier, the random number r_A received in step (A1), the new random number r_B (optional), the encrypted block *BE*, and the optional data field Text3. Then entity *B* signs the token data block with its private signature transformation S_B, appends optional Text4, and sends the resulting key token $KT_{B1} = S_B(A||r_A||r_B||BE||\text{Text3})||\text{Text4}$ to entity *A*.

Key Token Verification (A2.1) Entity *A* uses the sender's public verification transformation V_B to verify the digital signature in the received key token KT_{B1}. Then entity *A* checks its distinguishing identifier in KT_{B1} and checks that the received value r_A agrees with the random number sent in step (A1).

Key Block Decryption (A2.2) Entity *A* decrypts the block *BE* with its private decryption transformation D_A. Entity *A* then validates the sender's distinguishing identifier in *BE*. If all checks are successful, entity *A* accepts the key *K*.

NOTE 1 The number of protocol passes is 2.

NOTE 2 This mechanism provides implicit key authentication from entity *A* to entity *B*.

NOTE 3 This mechanism provides key confirmation from entity *B* to entity *A*. Entity *A* can be sure that it shares the correct key *K* with entity *B*, but entity *B* can only be sure that entity *A* has indeed received the key after it has obtained a secured message from entity *A* which has been processed using *K*.

NOTE 4 Entity *B* can choose the key.

NOTE 5 The tokens KT_{A1} and KT_{B1} conform to the tokens sent in the two-pass authentication mechanism described in 5.1.2 of ISO/IEC 9798-3[6] (note that the roles of entities *A* and *B* are exchanged). The token KT_{B1} accommodates the transfer of the key *K* through use of the optional data field: Text2 in the ISO/IEC 9798-3 mechanism has been replaced by *BE* || Text3.

NOTE 6 If this secret key transport mechanism is executed twice in parallel between two entities, then the resulting mutual secret key transport mechanism is in conformance with the mechanism described in ISO/IEC 9798-3.[6]

NOTE 7 Data field r_B is included for consistency with ISO/IEC 9798-3.[6] Because of the presence of *BE* in KT_{B1}, r_B is no longer required and is therefore optional in this mechanism.

12.5 Secret key transport mechanism 5

This secret key transport mechanism is based on the three-pass authentication mechanism of ISO/IEC 9798-3[6] and transfers two shared secret keys with mutual entity authentication and key confirmation. One key is transferred from entity *A* to entity *B* and one key from entity *B* to entity *A*. The following requirements shall be satisfied:

— Each entity *X* has an asymmetric signature system (S_X, V_X).

— Each entity *X* has an asymmetric encryption system (E_X, D_X).

— Each entity has access to an authenticated copy of the public verification transformation of the other entity. This may be achieved using the mechanisms described in Clause 13.

— Each entity has access to an authenticated copy of the public encryption transformation of the other entity. This may be achieved using the mechanisms described in Clause 13.

Secret key transport mechanism 5 is summarised in Figure 14.

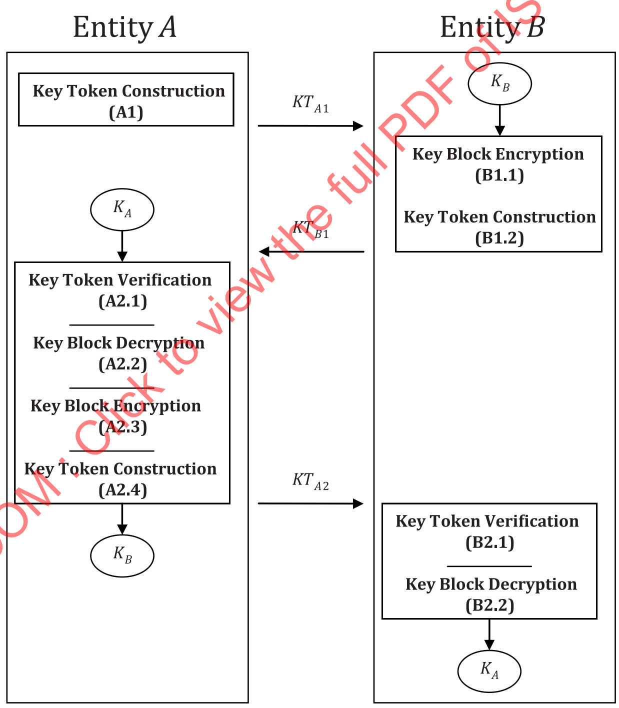

Figure 14 — Secret Key Transport Mechanism 5

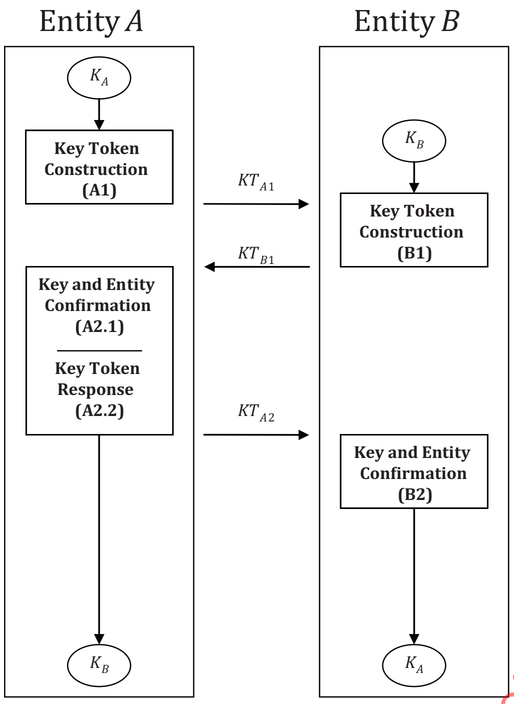

Figure 15 — Secret Key Transport Mechanism 6

Key Token Construction (A1) Entity *A* randomly generates r_A, constructs the key token $KT_{A1} = r_A||\text{Text1}$ and sends it to entity *B*.

Key Block Encryption (B1.1) Suppose *K* is a secret key that entity *B* wishes to securely transfer to entity *A*. Entity *B* constructs a block containing its own distinguishing identifier, the key K_B, and optional Text2, and encrypts the block using the recipient's public encryption transformation E_A:

$$BE_1 = E_A(B||K_B||\text{Text2}).$$

Key Token Construction (B1.2) Entity *B* randomly generates r_B and constructs a data block containing r_B, r_A, the recipient's identity, the encrypted key block BE_1, and optional Text3. Entity *B* signs the block using its private signature transformation S_B, appends optional Text4, and sends the key token $KT_{B1} = S_B(r_B||r_A||A||BE_1||\text{Text3})||\text{Text4}$ to entity *A*.

Key Token Verification (A2.1) Entity *A* verifies entity *B*'s signature on the key token KT_{B1} using entity *B*'s public verification transformation V_B, checks its distinguishing identifier in KT_{B1} and checks that the received value r_A agrees with the random number sent in step (A1).

Key Block Decryption (A2.2) Entity *A* decrypts the encrypted block BE_1 using its private decryption transformation D_A and checks the distinguishing identifier for entity *B*. If all checks are successful, entity *A* accepts the key K_B.

Key Block Encryption (A2.3) Entity *A* constructs a data block containing its own distinguishing identifier, its own key K_A, and optional Text5, and encrypts the block using the recipient's public encryption transformation E_B to obtain $BE_2 = E_B(A||K_A||\text{Text5})$.

Key Token Construction (A2.4) Entity *A* constructs a data block containing the random number r_A, the random number r_B, the recipient's distinguishing identifier, the encrypted key block BE_2, and optional Text6. Entity *A* signs the data block using its private signature transformation S_A, appends optional Text7, and sends the key token $KT_{A2} = S_A(r_A||r_B||B||BE_2||\text{Text6})||\text{Text7}$ to entity *B*.

Key Token Verification (B2.1) Entity *B* verifies entity *A*'s signature on the key token KT_{A2} using entity *A*'s public verification transformation V_A, checks its distinguishing identifier in KT_{A2} and checks that the received value r_B agrees with the random number sent in step (B1.2). In addition, B checks that the received value r_A agrees with the value contained in KT_{A1}.

Key Block Decryption (B2.2) Entity *B* decrypts the encrypted block BE_2 using its private decryption transformation D_B and verifies the distinguishing identifier for entity *A*. If all checks are successful, entity *B* accepts the key K_A. If only unilateral key transport is required then, as appropriate, either BE_1 or BE_2 can be omitted.

NOTE 1 The number of passes is 3.

NOTE 2 This mechanism provides mutual entity authentication, implicit key authentication of K_A from entity *B* to entity *A* and implicit key authentication of K_B from entity *A* to entity *B*.

NOTE 3 This mechanism provides key confirmation from sender to recipient for both keys K_A and K_B. Moreover, if entity *A* includes a MAC on K_B in the data field Text6 of KT_{A2}, then this mechanism provides mutual key confirmation with respect to K_B.

NOTE 4 Entity *A* can choose the key K_A, since it is the originating entity. Similarly, entity *B* can choose the key K_B. Joint key control can be achieved by each entity by combining the two keys K_A and K_B to form a shared secret key K_{AB}. The combination function shall be one-way, otherwise entity *A* can choose the shared secret key. This mechanism can then be classified as a key agreement mechanism.

NOTE 5 KT_{A1}, KT_{B1}, and KT_{A2} are compatible with the tokens sent in the three pass authentication mechanism described in Clause 5.2.2 of ISO/IEC 9798-3.[6] The second token accommodates the transfer of the key K_B: Text2 of the ISO/IEC 9798-3 mechanism has been replaced by BE_1||Text3. The third token accommodates the transfer of the key K_A: Text4 of the ISO/IEC 9798-3 mechanism has been replaced by BE_2||Text6. The third token can also accommodate the transfer of a MAC within Text6.

NOTE 6 If the data fields Text1 and Text4 (or Text7 and Text4) contain the public key certificates of entities *A* and *B*, respectively, then the third and fourth requirements at the beginning of 12.5 can be relaxed to the requirement that both entities are in possession of an authenticated copy of the CA's public verification key.

12.6 Secret key transport mechanism 6

This secret key transport mechanism securely transfers two secret keys in three passes, one from entity *A* to entity *B* and one from entity *B* to entity *A*. In addition, the mechanism provides mutual entity authentication. This mechanism is based on the following requirements:

— Each entity *X* has an asymmetric encryption system (E_X, D_X).

— Each entity has access to an authenticated copy of the public encryption transformation of the other entity. This may be achieved using the mechanisms described in Clause 13.

Secret key transport mechanism 6 is summarised in Figure 15.

Key Token Construction (A1) Entity *A* has obtained a key K_A and wants to transfer it securely to entity *B*. Entity *A* selects a random number r_A and constructs a key data block consisting of its distinguishing identifier, the key K_A, the number r_A and an optional data field Text1. Then entity *A* encrypts the key block using entity B's public encryption transformation E_B, thereby producing the encrypted data block $BE_1 = E_B(A||K_A||r_A||\text{Text1})$.

Entity *A* constructs the token $KT_{A1} = BE_1$||Text2, consisting of the encrypted data block and some optional data field Text2.

Entity *A* sends the token to entity *B*.

Key Token Construction (B1) Entity *B* extracts the encrypted key block BE_1 from the received key token KT_{A1} and decrypts it using its private decryption transformation D_B. Then entity *B* checks that the decrypted version of BE_1 contains the identifier for entity *A*.

Entity *B* has obtained a key K_B and wants to transfer it securely to entity *A*. Entity *B* selects a random number r_B and constructs a key data block consisting of the distinguishing identifier for entity *B*, the key K_B, the random number r_B, the random number r_A (as extracted from the decrypted block) and an optional data field Text3. Then entity B encrypts the key block using entity A's public encryption transformation E_A, thereby producing the encrypted data block $BE_2 = E_A(B||K_B||r_A||r_B||\text{Text3})$.

Then entity B constructs the key token $KT_{B1} = BE_2||\text{Text4}$, consisting of the encrypted data block BE_2 and an optional data field Text4.

Entity B sends the token to entity A.

Key and Entity Confirmation (A2.1) Entity A extracts the encrypted key block BE_2 from the received key token KT_{B1} and decrypts it using its private decryption transformation D_A. Then entity A checks the validity of the key token through comparison of the random number r_A with the random number r_A contained in the encrypted block BE_2. If the verification is successful, entity A has implicitly confirmed that K_A has safely reached entity B.

Key Token Response (A2.2) Entity A extracts the random number r_B from the decrypted key block and constructs the key token $KT_{A2} = r_B||\text{Text5}$, consisting of the random number r_B and an optional data field Text5.

Entity A sends the token to entity B.

Key and Entity Confirmation (B2) Entity B verifies that the response r_B extracted from KT_{A2} is consistent with the random number r_B sent in encrypted form in KT_{B1}. If the verification is successful, entity B has authenticated entity A and at the same time has obtained confirmation that K_B has safely reached entity A.

NOTE 1 The number of passes is 3.

NOTE 2 This mechanism provides implicit key authentication of K_A from entity B to entity A and implicit key authentication of K_B from entity A to entity B.

NOTE 3 Entity A can choose the key K_A, since it is the originating entity. Similarly, entity B can choose the key K_B. Joint key control can be achieved by each entity by combining the two keys K_A and K_B on both sides to form a shared secret key K_{AB}. However, the combination function shall be one-way, otherwise entity B can choose the shared secret key. This mechanism could then be classified as a key agreement mechanism.

NOTE 4 This mechanism uses asymmetric techniques to mutually transfer two secret keys, K_A from entity A to entity B and K_B from entity B to entity A. The following cryptographic function separation can be derived from the mechanism: entity A uses its key K_A to encrypt messages for entity B and to verify authentication codes from entity B. Entity B in turn uses the received key K_A to decrypt messages from entity A and generate authentication codes for entity A. The cryptographic functions of K_B can be separated in an analogous manner. In such a way, the asymmetric basis of the key transport mechanism can be extended to the usage of the secret keys.

NOTE 5 This mechanism is derived from the three pass protocol known as COMSET.[18]

NOTE 6 This mechanism is based on zero-knowledge techniques. From the execution of the mechanism, neither of the entities learns anything that it could not have computed itself.

13 Public key transport

13.1 Public key transport mechanism 1

If entity A has access to a protected channel (i.e., a channel which provides data origin authentication and data integrity), such as a courier, registered mail, etc., to entity B then entity A may transport its public key information directly via that protected channel to entity B. This is the most elementary form of transferring a public key. The following requirements shall be satisfied:

— Entity A's public key information PKI_A contains at least entity A's distinguishing identifier and entity A's public key. In addition it may contain a serial number, a validity period, a time stamp and other data elements.

— Since the public key information does not contain any secret data, the communication channel need not provide confidentiality.

Public key transport mechanism 1 is summarised in Figure 16.

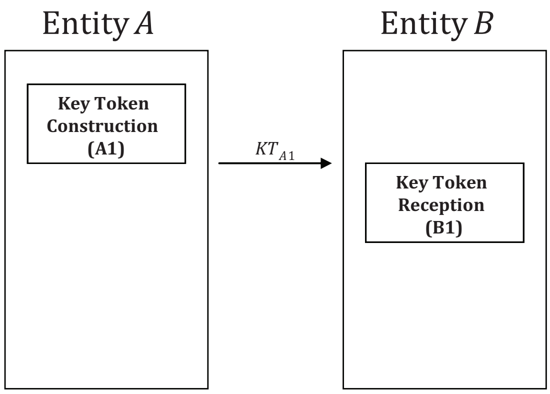

Figure 16 — Public Key Transport Mechanism 1

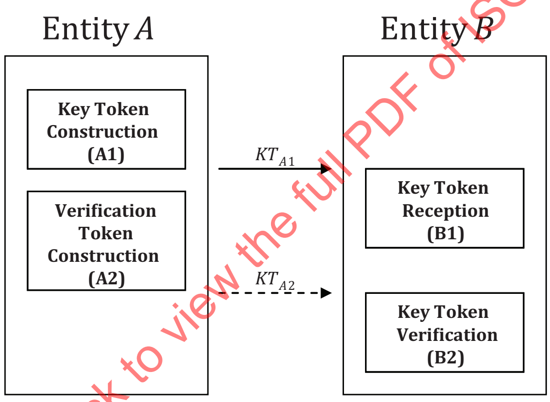

Figure 17 — Public Key Transport Mechanism 2

Key Token Construction (A1) Entity *A* constructs the key token KT_{A1} containing the public key information of entity *A* and some optional data field Text, and sends $KT_{A1} = PKI_A$||Text via a protected channel to entity *B*.

Key Token Reception (B1) Entity *B* receives the key token via the protected channel from entity *A*, retrieves entity *A*'s public key information PKI_A and stores entity *A*'s public key into the list of active public keys (this list shall be protected from tampering).

NOTE 1 This mechanism can be used to transfer public verification keys (for an asymmetric signature system) or public encryption keys (for an asymmetric encryption system) or public key agreement keys.

NOTE 2 Authentication in this context includes both data integrity and data origin authentication (as defined in ISO 7498-2[1]).

13.2 Public key transport mechanism 2

This mechanism transports the public key information of entity *A* via an unprotected channel to entity *B*. To verify the integrity and the origin of the received public key information a second authenticated channel is used. Such a mechanism is useful when the public key information *PKI* is transferred electronically on a high bandwidth channel, whereas the authentication of the public key information takes place over an authenticated low bandwidth channel such as a telephone, courier, or registered mail. As an additional requirement, the entities shall share a common hash, as defined in ISO/IEC 10118-1. The following requirements shall be satisfied:

— Entity *A*'s public key information PKI_A contains at least entity *A*'s distinguishing identifier and entity *A*'s public key. In addition it may contain a serial number, a validity period, a time stamp and other data elements.

— Since the public key information does not contain any secret data, the communication channel need not provide confidentiality.

Public key transport mechanism 2 is summarised in Figure 17.

Key Token Construction (A1) Entity *A* constructs the key token KT_{A1} containing the public key information of entity *A* and sends $KT_{A1} = PKI_A||\text{Text1}$ to entity *B*.

Key Token Reception (B1) Entity *B* receives the key token, retrieves entity *A*'s public key information PKI_A, and stores it protected from tampering for later verification and use.

Verification Token Construction (A2) Entity *A* computes a check value hash(PKI_A) on its public key information and sends this check value together with the optional distinguishing identifiers of entities *A* and *B* to entity *B* using a second independent and authenticated channel (e.g. a courier or registered mail), where

$$KT_{A2} = \text{A}||\text{B}||\text{hash}(PKI_A)||\text{Text2}.$$

Key Token Verification (B2) Upon reception of the verification token KT_{A2}, *B* optionally checks the distinguishing identifier of entities *A* and *B*, computes the check value on the public key information of entity *A* received in the key token KT_{A1} and compares it with the check value received in the verification token KT_{A2}. If the check succeeds, entity *B* puts entity *A*'s public key onto the list of active public keys (this list shall be protected from tampering).

NOTE 1 This mechanism can be used to transfer public verification keys (for an asymmetric signature system) or public encryption keys (for an asymmetric encryption system) or public key agreement keys.

NOTE 2 Authentication in this context includes both data integrity and data origin authentication.

NOTE 3 If the public key that is transported is a key for an asymmetric signature system not giving message recovery, then entity *A* can sign the token KT_{A1} using the corresponding private signature key. In that case, the verification of entity *A*'s signature in step (B1) using the received public verification key confirms that entity *A* knew the corresponding private signature key, and so presumably, was the only entity that knew the corresponding private signature key at the time the token was created. If a time stamp is used in PKI_A, then verification confirms that entity *A* currently knows the corresponding private signature key.

NOTE 4 A manually signed letter from Entity *A* can be used for the verification token.

13.3 Public key transport mechanism 3

This mechanism transfers a public key from entity *A* to entity *B* in an authenticated way by using a trusted third party. The authentication of the entities' public keys can be ensured by exchanging the public keys in the form of public key certificates. Entity *A*'s public key certificate contains the public key information, together with a digital signature provided by a trusted third party, the Certification Authority (CA). The introduction of a CA reduces the problem of authenticated user public key distribution to the problem of authenticated distribution of the CA's public key, at the expense of a trusted centre (the CA). 11770-1 shall be referred. See also ISO/IEC 9594-8[2] (Annex E).

This mechanism is based on the assumption that a valid public key certificate $Cert_A$ of entity *A*'s public key information PKI_A has been issued by some certification authority, and that entity *B* has access to an authenticated copy of the public verification transformation V_{CA} of that certification authority CA which has issued the public key certificate.

Public key transport mechanism 3 is summarised in Figure 18.

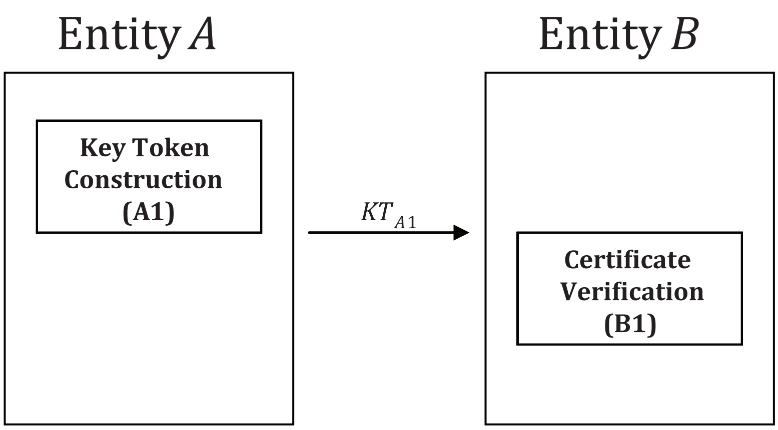

Figure 18 — Public Key Transport Mechanism 3

Key Token Construction (A1) Entity *A* constructs the key token KT_{A1} containing the public key certificate of entity *A* and sends it to entity *B*, $KT_{A1} = Cert_A||\text{Text}$.

Certificate Verification (B1) Upon reception of the public key certificate, entity *B* uses the public verification transformation V_{CA} of the certification authority to verify the authenticity of the public key information and to check the validity of entity *A*'s public key.

If entity *B* wants to make sure that entity *A*'s public key certificate has not been revoked recently, then entity *B* should consult a trusted third party (such as the CA) via some authenticated channel.

NOTE 1 The number of passes is 1, but there could have been a request from entity *B* to entity *A* for the transfer of the public key certificate. This additional pass is optional and not shown here. Entity *A*'s public key certificate could also be distributed by a directory, in which case this public key transport mechanism would be executed between the directory and entity *B*.

NOTE 2 Entity authentication is not provided by this mechanism.

NOTE 3 Receiving a public key certificate provides confirmation that the public key has been certified by the CA.

NOTE 4 The public verification key v_{CA} of the CA shall be made available to entity *B* in an authenticated way. This can be done using the mechanisms described in Clause 13.

Annex A
(normative)

Object identifiers

This annex lists the object identifiers assigned to the key management mechanisms specified in this part of ISO/IEC 11770.

```
Key-management-AsymmetricTechniques {

iso(1) standard(0) key-management(11770)

asymmetricTechniques(3) asn1-module(0) object-identifiers(0) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

-- IMPORTS None; --

OID ::= OBJECT IDENTIFIER – Alias

-- Synonyms –

id-km-at OID ::= {

iso(1) standard(0) key-management(11770) asymmetricTechniques(3) }

-- Assignments –

id-km-at-kAM-1 OID ::= { id-km-at keyAgreementMechanism1(1) }

id-km-at-kAM-2 OID ::= { id-km-at keyAgreementMechanism2(2) }

id-km-at-kAM-3 OID ::= { id-km-at keyAgreementMechanism3(3) }

id-km-at-kAM-4 OID ::= { id-km-at keyAgreementMechanism4(4) }

id-km-at-kAM-5 OID ::= { id-km-at keyAgreementMechanism5(5) }

id-km-at-kAM-6 OID ::= { id-km-at keyAgreementMechanism6(6) }

id-km-at-kAM-7 OID ::= { id-km-at keyAgreementMechanism7(7) }

id-km-at-kAM-8 OID ::= { id-km-at keyAgreementMechanism8(8) }

id-km-at-kAM-9 OID ::= { id-km-at keyAgreementMechanism9(9) }

id-km-at-kAM-10 OID ::= { id-km-at keyAgreementMechanism10(10) }

id-km-at-kAM-11 OID ::= { id-km-at keyAgreementMechanism11(11) }

id-km-at-kAM-12 OID ::= { id-km-at keyAgreementMechanism12(21) }

id-km-at-kTM-1 OID ::= { id-km-at keyTransportMechanism1(12) }

id-km-at-kTM-2 OID ::= { id-km-at keyTransportMechanism2(13) }

id-km-at-kTM-3 OID ::= { id-km-at keyTransportMechanism3(14) }

id-km-at-kTM-4 OID ::= { id-km-at keyTransportMechanism4(15) }

id-km-at-kTM-5 OID ::= { id-km-at keyTransportMechanism5(16) }

id-km-at-kTM-6 OID ::= { id-km-at keyTransportMechanism6(17) }
```

```
id-km-at-pKT-1 OID ::= { id-km-at publicKeyTransportMechanism1(18) }

id-km-at-pKT-2 OID ::= { id-km-at publicKeyTransportMechanism2(19) }

id-km-at-pKT-3 OID ::= { id-km-at publicKeyTransportMechanism3(20) }

-- Key Agreement Mechanism 1 –

keyConstruction-1a OID ::= {

   id-km-at-kAM-1 keyConstructionFunction-1a(1) }

keyConstruction-1b OID ::= {

   id-km-at-kAM-1 keyConstructionFunction-1b(2) }

-- Key Agreement Mechanism 2 –

keyTokenConstruction-2 OID ::= {

   id-km-at-kAM-2 keyTokenConstructionFunction(1) }

keyConstruction-2a OID ::= {

   id-km-at-kAM-2 keyConstructionFunction-2a(2) }

keyConstruction-2b OID ::= {

   id-km-at-kAM-2 keyConstructionFunction-2b(3) }

-- Key Agreement Mechanism 3 –

keyConstruction-3a OID ::= {

   id-km-at-kAM-3 keyConstructionFunction-3a(1) }

keyTokenSignature-3 OID ::= {

   id-km-at-kAM-3 keyTokenSignatureFunction(2) }

keyConstruction-3b OID ::= {

   id-km-at-kAM-3 keyConstructionFunction-3b(3) }

signatureVerification-3 OID ::= {

   id-km-at-kAM-3 signatureVerificationFunction(4) }

-- Key Agreement Mechanism 4 –

keyTokenConstruction-4a OID ::= {

   id-km-at-kAM-4 keyTokenConstructionFunction-4a(1) }

keyTokenConstruction-4b OID ::= {

   id-km-at-kAM-4 keyTokenConstructionFunction-4b(2) }

keyConstruction-4a OID ::= {

   id-km-at-kAM-4 keyConstructionFunction-4a(3) }

keyConstruction-4b OID ::= {

   id-km-at-kAM-4 keyConstructionFunction-4b(4) }

-- Key Agreement Mechanism 5 –

keyTokenConstruction-5a OID ::= {
```

```
   id-km-at-kAM-5 keyTokenConstructionFunction-5a(1) }
keyTokenConstruction-5b OID ::= {
   id-km-at-kAM-5 keyTokenConstructionFunction-5b(2) }
keyConstruction-5a OID ::= {
   id-km-at-kAM-5 keyConstructionFunction-5a(3) }
keyConstruction-5b OID ::= {
   id-km-at-kAM-5 keyConstructionFunction-5b(4) }
-- Key Agreement Mechanism 6 –
keyTokenConstruction-6 OID ::= {
   id-km-at-kAM-6 keyTokenConstructionFunction(1) }
keyTokenProcessing-6b OID ::= {
   id-km-at-kAM-6 keyTokenProcessingFunction-6b(2) }
keyConstruction-6 OID ::= {
   id-km-at-kAM-6 keyConstructionFunction(3) }
keyTokenProcessing-6a OID ::= {
   id-km-at-kAM-6 keyTokenProcessingFunction-6a(4) }
-- Key Agreement Mechanism 7 –
keyTokenConstruction-7 OID ::= {
   id-km-at-kAM-7 keyTokenConstructionFunction(1) }
keyTokenProcessingAndKeyConstruction-7 OID ::= {
   id-km-at-kAM-7 keyTokenProcessingAndKeyConstructionFunction(2) }
keyTokenProcessing-7a OID ::= {
   id-km-at-kAM-7 keyTokenProcessingFunction-7a(4) }
keyTokenProcessing-7b OID ::= {
   id-km-at-kAM-7 keyTokenProcessingFunction-7b(5) }
-- Key Agreement Mechanism 8 –
keyTokenConstruction-8 OID ::= {
   id-km-at-kAM-8 keyTokenConstructionFunction(1) }
keyConstruction-8a OID ::= {
   id-km-at-kAM-8 keyConstructionFunction-8a(2) }
keyConstruction-8b OID ::= {
   id-km-at-kAM-8 keyConstructionFunction-8b(3) }
-- Key Agreement Mechanism 9 –
keyTokenConstruction-9a OID ::= {
   id-km-at-kAM-9 keyTokenConstructionFunction-9a(1) }
keyTokenConstruction-9b OID ::= {
```

```
   id-km-at-kAM-9 keyTokenConstructionFunction-9b(2) }
keyConstruction-9a OID ::= {
   id-km-at-kAM-9 keyConstructionFunction-9a(3) }
keyConstruction-9b OID ::= {
   id-km-at-kAM-9 keyConstructionFunction-9b(4) }
-- Key Agreement Mechanism 10 –
keyTokenConstruction-10a OID ::= {
   id-km-at-kAM-10 keyTokenConstructionFunction(1) }
keyConstruction-10b OID ::= {
   id-km-at-kAM-10 keyConstructionFunction-10b(2) }
keyConstruction-10a OID ::= {
   id-km-at-kAM-10 keyConstructionFunction-10a(3) }
verification-10b OID ::= {
   id-km-at-kAM-10 verificationFunction(4) }
-- Key Agreement Mechanism 11 –
entityConfirmation-11a OID ::= {
   id-km-at-kAM-11 entityConfirmationFunction-11a(1) }
entityConfirmation-11b OID ::= {
   id-km-at-kAM-11 entityConfirmationFunction-11b(2) }
keyTokenAndKeyConstruction-11 OID ::= {
   id-km-at-kAM-11 keyTokenProcessingAndKeyConstructionFunction(3) }
keyConstruction-11 OID ::= {
   id-km-at-kAM-11 keyConstructionFunction(4) }
keyVerification-11 OID ::= {
   id-km-at-kAM-11 keyVerificationFunction(5) }
-- Key Transport Mechanism 1 –
keyTokenConstruction-1 OID ::= {
   id-km-at-kTM-1 keyTokenConstructionFunction(1) }
keyTokenDeconstruction-1 OID ::= {
   id-km-at-kTM-1 keyTokenDeconstructionFunction(2) }
-- Key Transport Mechanism 2 –
keyEncryption-2 OID ::= {
   id-km-at-kTM-2 keyEncryptionFunction(1) }
keyTokenConstruction-2a OID ::= {
   id-km-at-kTM-2 keyTokenConstructionFunction(2) }
```

```
keyTokenVerification-2 OID ::= {
   id-km-at-kTM-2 keyTokenVerificationFunction(3) }
keyDecryption-2 OID ::= {
   id-km-at-kTM-2 keyDecryptionFunction(4) }
-- Key Transport Mechanism 3 –
keyBlockSignature-3 OID ::= {
   id-km-at-kTM-3 keyBlockSignatureFunction(1) }
keyTokenConstruction-3 OID ::= {
   id-km-at-kTM-3 keyTokenConstructionFunction(2) }
keyTokenDecryption-3 OID ::= {
   id-km-at-kTM-3 keyTokenDecryptionFunction(3) }
keyBlockVerification-3 OID ::= {
   id-km-at-kTM-3 keyBlockVerificationFunction(4) }
-- Key Transport Mechanism 4 –
keyTokenConstruction-4c OID ::= {
   id-km-at-kTM-4 keyTokenConstructionFunction-4c(1) }
keyBlockEncryption-4 OID ::= {
   id-km-at-kTM-4 keyBlockEncryptionFunction(2) }
keyTokenConstruction-4d OID ::= {
   id-km-at-kTM-4 keyTokenConstructionFunction-4d(3) }
keyTokenVerification-4 OID ::= {
   id-km-at-kTM-4 keyTokenVerificationFunction(4) }
keyBlockDecryption-4 OID ::= {
   id-km-at-kTM-4 keyBlockDecryptionFunction(5) }
-- Key Transport Mechanism 5 –
keyTokenConstruction-5c OID ::= {
   id-km-at-kTM-5 keyTokenConstructionFunction-5c(1) }
keyBlockEncryption-5b OID ::= {
   id-km-at-kTM-5 keyBlockEncryptionFunction-5b(2) }
keyTokenConstruction-5d OID ::= {
   id-km-at-kTM-5 keyTokenConstructionFunction-5d(3) }
keyTokenVerification-5a OID ::= {
   id-km-at-kTM-5 keyTokenVerificationFunction-5a(4) }
keyBlockDecryption-5a OID ::= {
   id-km-at-kTM-5 keyBlockDecryptionFunction-5a(5) }
keyBlockEncryption-5a OID ::= {
```

```
   id-km-at-kTM-5 keyBlockEncryptionFunction-5a(6) }
keyTokenConstruction-5e OID ::= {
   id-km-at-kTM-5 keyTokenConstructionFunction-5e(7) }
keyTokenVerification-5b OID ::= {
   id-km-at-kTM-5 keyTokenVerificationFunction-5b(8) }
keyBlockDecryption-5b OID ::= {
   id-km-at-kTM-5 keyBlockDecryptionFunction-5b(9) }
-- Key Transport Mechanism 6 –
keyTokenConstruction-6a OID ::= {
   id-km-at-kTM-6 keyTokenConstructionFunction-6a(1) }
keyTokenConstruction-6b OID ::= {
   id-km-at-kTM-6 keyTokenConstructionFunction-6b(2) }
keyEntityConfirmation-6a OID ::= {
   id-km-at-kTM-6 keyEntityConfirmationFunction-6a(3) }
keyTokenResponse-6 OID ::= {
   id-km-at-kTM-6 keyResponseFunction(4) }
keyEntityConfirmation-6b OID ::= {
   id-km-at-kTM-6 keyEntityConfirmationFunction-6b(5) }
-- Public Key Transport Mechanism 1 –
keyTokenConstruction-1a OID ::= {
   id-km-at-pKT-1 keyTokenConstructionFunction(1) }
keyTokenReception-1 OID ::= {
   id-km-at-pKT-1 keyTokenReceptionFunction(2) }
-- Public Key Transport Mechanism 2 –
keyTokenConstruction-2b OID ::= {
   id-km-at-pKT-2 keyTokenConstructionFunction(1) }
keyTokenReception-2 OID ::= {
   id-km-at-pKT-2 keyTokenReceptionFunction(2) }
keyTokenVerification-2a OID ::= {
   id-km-at-pKT-2 keyTokenVerificationFunction(3) }
-- Public Key Transport Mechanism 3 –
keyTokenConstruction-3a OID ::= {
   id-km-at-pKT-3 keyTokenConstructionFunction(1) }
certificationVerification-3 OID ::= {
   id-km-at-pKT-3 certificationVerificationFunction(2) }
```



```
END -- Key-management-AsymmetricTechniques --
```

Annex B
(informative)

Properties of key establishment mechanisms

The following tables summarize the major properties of the key establishment/transport mechanisms specified in this part of ISO/IEC 11770.

The following notation is used in Table B.1 — Properties of key agreement mechanisms, Table B.2 — Properties of secret key transport mechanisms, and Table B.3 — Properties of public key transport mechanisms:

A mechanism provides the property with respect to entity *A*.

B mechanism provides the property with respect to entity *B*.

A, B the mechanism provides the property with respect to both entities, *A* and *B*.

No the mechanism does not provide the property.

Opt the mechanism can provide the property as an option, using additional means.

(A) the mechanism can optionally provide the property with respect to entity *A*, using additional means.

(B) the mechanism can optionally provide the property with respect to entity *B*, using additional means.

MFS the mechanism provides mutual forward secrecy.

#passes the number of passes.

Public key operations in Tables B.1, B.2, and B.3: the number of computations of asymmetric transformation. F and FP, the number of computations of asymmetric transformation executed by entity X , E_X , D_X , S_X , and V_X . "(2F,1F)" means that entity *A* needs two computations of the function F and entity *B* needs one computation of the function F in Key Agreement Mechanism 2 in Table B. 1; and the number of computations of asymmetric transformation, "(1FP,1FP,1FP)" means that entity *A* needs one computation of the function FP, entity *B* needs one computation of the function FP, and entity *C* needs one computation of the function FP in Key Agreement Mechanism 12 in Table B. 1. "(1E_B ,1D_B)" means that entity *A* needs one computation of the function E_B and entity *B* needs one computation of the function D_B in Table B.2. "(0 ,1V_{CA})" means that entity *B* needs one computation of the public verification transformation V_{CA} of the certification authority CA in Table B.3.

Another important property that can be derived from key freshness is replay attack prevention. Replay attacks are generally not possible where key freshness is guaranteed for both entities.

The property of implicit key authentication has direction by its definition. When the table for implicit key authentication has an "A", this means that entity *B* is assured that entity *A* is the only other entity that can possibly be in possession of the correct key. When the table for implicit key authentication has an "A, B", this means that entities *A* and *B* are assured that only the other entity can possibly be in possession of the correct key.

NOTE 1 Only mechanism 12 in Table B.1 executes among three entities and others execute among two entities.

NOTE 2 All mechanisms except mechanism 1 in Table B.1 use secure random bit generation.

Table B.1 — Properties of key agreement mechanisms

Mechanism	#passes	Implicit key authentication	Key confirmation	Entity authentication	Public key operations	Forward secrecy	Key freshness
1	0	A, B	No	No	(1F, 1F)	No	No
2	1	B	No	No	(2F, 1F)	A	A
3	1	A, B	B	A	(2F/1S$_A$, 1F/1V$_A$)	A	A
4	2	No	No	No	(2F, 2F)	MFS	A, B
5	2	A, B	Opt	No	(3F, 3F)	A,B	A, B
6	2	A, B	Opt	B	(1V$_B$/1D$_A$, 1S$_B$/1E$_A$)	B	A, B
7	3	A, B	A, B	A, B	(2F/1V$_B$/1S$_A$, 2F/1S$_B$/1V$_A$)	MFS	A, B
8	1	A, B	No	No	(2F, 1F)	A	A
9	2	A, B	No	No	(2F, 2F)	MFS	A, B
10	3	A, B	A, B	A, B	(2F, 2F)	MFS	A, B
11	4	B	A, B	B	(1V_{CA}/1E$_B$, 1D$_B$)	MFS	A, B
12	0	A, B, C	No	No	(1FP, 1FP, 1FP)	No	No
F.3	2	A, B	No	No	(3F+2FP, 3F+2FP)	A, B	A, B
F.4	2	A, B	No	No	(3F+2FP, 3F+2FP)	A, B	A, B

Table B.2 — Properties of secret key transport mechanisms

Mechanism	#passes	Implicit key authentication	Key confirmation	Key control	Entity authentication	Public key operations	Forward secrecy	Key freshness
1	1	B	No	A	No	(1E$_B$, 1D$_B$)	A	A
2	1	B	B	A	A	(1E$_B$/1S$_A$, 1V$_A$/1D$_B$)	A	A
3	1	B	B	A	A	(1S$_A$/1E$_B$, 1D$_B$/1V$_A$)	A	A
4	2	A	A	B	B	(1V$_B$/1D$_A$, 1E$_A$/1S$_B$)	B	A
5	3	A, B	(A), B	A, B	A, B	(1V$_B$/1D$_A$ /1E$_B$/1S$_A$, 1E$_A$/1S$_B$ /1V$_A$/1D$_B$)	No	A, B
6	3	A, B	No	A, B	No	(1E$_B$/1D$_A$, 1D$_B$/1E$_A$)	No	A, B

Table B.3 — Properties of public key transport mechanisms

Mechanism	#passes	Implicit key authentication	Key confirmation	Key control	Entity authentication	Public key operations	Forward secrecy	Key freshness
1	1	-	-	A	A	(0, 0)	-	No
2	2	-	-	A	A	(0, 0)	-	No
3	1	-	-	A	A	(0, 1V_{CA})	-	No

Annex C
(informative)

Examples of key derivation functions

C.1 ASN.1 syntax for key derivation functions

This clause describes ASN.1 syntax for a key derivation function.

The input to the key derivation function is the shared secret *ZZ* and other information OtherInfo.The other information includes the initiator's information entityAInfo, and the responder's information entityBInfo, suppPubInfo, and suppPrivInfo.

OtherInfo ::= SEQUENCE {

keyInfo KeySpecificInfo,

entityAInfo [0] OCTET STRING OPTIONAL,

entityBInfo[1] OCTET STRING OPTIONAL,

suppPubInfo[2] OCTET STRING OPTIONAL,

suppPrivInfo[3] OCTET STRING OPTIONAL

}

KeySpecificInfo ::= SEQUENCE {

algorithm OBJECT IDENTIFIER,

counter Counter

}

Counter ::= INTEGER (1...32767)

The suppPubInfo and suppPrivInfo fields are optional fields used in key derivation. These fields may be used to hold additional, supplementary public and private information that is mutually known to the communicating parties, but that is not specific to either party.

The contents of suppPubInfo and suppPrivInfo are defined by the key management protocol. The definition, syntax, and encoding rules of the suppPubInfo and suppPrivInfo fields are the responsibility of the key management protocol and are beyond the scope of this part of ISO/IEC 11770.

All inputs to the key derivation hash function shall be an integral number of octets in length. suppPrivInfo may include *ZZ*.

NOTE 1 Some mechanisms in Clauses 11 and 12 derive shared secrets either as points on the elliptic curve or as the concatenation of two points on an elliptic curve. In the first situation, in order to obtain a shared secret integer z for input into the key derivation function, the function π should be applied to the point.

NOTE 2 OtherInfo is used in Annexes C.3, C.5, and C.6.

C.2 The IEEE P1363 key derivation function

This clause describes the key derivation function that is given in the IEEE P1363 standard.[14]

Preconditions As a precondition of the use of this key derivation function, users shall agree on a common hash function. Users who use different hash functions will obtain different results. For the purposes of this part of ISO/IEC 11770, the hash function is referred in ISO/IEC 10118. The shared key that is produced will have length equal to the length of the output of the hash function.

Input The inputs to this key derivation function are

— The shared secret z which is an integer, expressed as an octet string.

— The key derivation parameters, *parameters*, also expressed as an octet string.

NOTE 1 Users shall also agree on a common method of converting integers and parameters to octet strings for input into the key derivation function.

Actions If the combined length of the shared secret z and the parameters exceeds any limitation that may exist for the agreed hash function, hash, then output "error" and stop.

Otherwise compute the value $K = \text{hash}(z \,||\, parameters)$.

Output Output K as the key.

C.3 The ANSI X9.42 key derivation function

This element describes a key derivation function based on the key derivation function that is given in the ANSI X9.42 standard.[12]

Prerequisites A hash function specified in ISO/IEC 10118 is chosen. Let *hashlen* denote the length of the output of the hash function chosen, and let *maxhashlen* denote the maximum length of the input to the hash function.

Input The input to the key derivation function is:

— *ZZ*: A bit string denoting the shared secret.

NOTE 1 Some mechanisms in Clauses 11 and 12 derive shared keys K_{AB} either as points on the elliptic curve or as the concatenation of two points on an elliptic curve. In the first situation, in order to obtain a shared secret value *ZZ* for input into the key derivation function, the function π should be applied to the point and the resulting integer converted to a bit string. In the second situation, the function π should be applied to both points to obtain two integers z_1 and z_2. The two integers should then be converted to bit strings and concatenated (or combined using any prefix-free encoding method), as were the points, to obtain the appropriate bit string.

— *keydatalen*: An integer representing the length in bits of the keying data to be generated. This integer is less than ($hashlen \times (2^{32}–1)$).

— OtherInfo: A bit string, specified in ASN.1 DER encoding, consisting of the following key specification information as specified in Annex C.2

— AlgorithmID: a unique object identifier (OID) of the symmetric algorithm(s) with which the keying data will be used.

— Counter: a 32-bit octet string, with initial value 00000001 (in hexadecimal).

— (Optional) EntityAInfo: A bit string containing public information contributed by the initiator.

— (Optional) EntityBInfo: A bit string containing public information contributed by the responder.

— (Optional) SuppPrivInfo: A bit string containing some additional, mutually known private information, e.g., a shared secret symmetric key communicated through a separate channel.

— (Optional) SuppPubInfo: A bit string containing some additional, mutually known public information.

NOTE 2 Users shall also agree on a common method of converting integers and parameters to bit strings for input into the key derivation function.

Actions The key derivation function is computed as follows:

a) Let $d = \lceil keydatalen / hashlen \rceil$.

b) Initialize Counter = 00000001 (in hexadecimal).

c) For i = 1 to d,

— Compute h_i = hash(ZZ || OtherInfo) where h_i denotes the hash value computed using the appropriate hash function, and OtherInfo = AlgorithmID || Counter [|| EntityAInfo || EntityBInfo || SuppPrivInfo || SuppPubInfo].

— Increment Counter.

— Increment i.

d) Compute K = leftmost *keydatalen* bits of $h_1 || h_2 || ... || h_d$.

e) Output K.

Output The keying data K as a bit string of length *keydatalen* bits.

Note that this key derivation function based on ASN.1 DER encoding produces keying data which is less than $hashlen\times(2^{32}-1)$ bits in length. It is assumed that all key derivation function calls are indeed for bit strings which are less than $hashlen\times(2^{32}-1)$ bits in length. Any scheme attempting to call the key derivation function using a bit string that is greater than or equal to $hashlen\times(2^{32}-1)$ bits shall output "invalid" and stop. Similarly, it is assumed that all key derivation function calls do not involve hashing a bit string that is more than *maxhashlen* bits in length. Any scheme attempting to call the key derivation function on a call involving hashing a bit string that is greater than *maxhashlen* bits shall output "invalid" and stop.

C.4 The ANSI X9.63 key derivation function

This clause describes a key derivation function based on the key derivation function that is given in the ANSI X9.63 standard.[13]

Prerequisites The prerequisite for the operation of the key derivation function is that a hash function, hash, specified in ISO/IEC 10118 is chosen. Let *hashlen* denote the length of the output of the hash function chosen, and let *maxhashlen* denote the maximum length of the input to the hash function.

Input The input to the key derivation function is:

— A bit string Z which is the shared secret.

NOTE 1 Some mechanisms in Clauses 11 and 12 derive shared keys K_{AB} either as points on the elliptic curve or as the concatenation of two points on an elliptic curve. In the first situation, in order to obtain a shared secret Z for input into the key derivation function, the function π should be applied to the point and the resulting integer converted to a bit string. In the second situation, the function π should be applied to both points to obtain two integers z_1 and z_2. The two integers should then be converted to bit strings and concatenated (or combined using any prefix-free encoding method), as were the points, to obtain the appropriate bit string.

— An integer *keydatalen* which is the length in bits of the keying data to be generated. *keydatalen* shall be less than $hashlen\times(2^{32}-1)$.

— (Optional) A bit string SharedInfo which consists of some data shared by the two entities intended to share the secret Z.

NOTE 2 Users shall also agree on a common method of converting integers and parameters to bit strings for input into the key derivation function.

Actions The key derivation function is computed as follows:

— Initiate a 32-bit, big-endian bit string counter as 00000001 (in hexadecimal).

— For $i = 1$ to $j = \lceil keydatalen / hashlen \rceil$, do the following:

— Compute $\text{Hash}_i = H(Z \,||\, \text{counter} \,[\, ||\, \text{SharedInfo}\,])$.

— Increment counter.

— Increment *i*.

— Let $H\text{Hash}_j$ denote Hash_j if *keydatalen*/*hashlen* is an integer, and let it denote the (*keydatalen* - (*hashlen*×(*j*-1))) leftmost bits of Hash_j otherwise.

— Set KeyData = $\text{Hash}_1 \,||\, \text{Hash}_2 \,||\, \ldots \,||\, \text{Hash}_{j-1} \,||\, H\text{Hash}_j$.

Output The bit string KeyData of length *keydatalen* bits.

Note that the key derivation function produces keying data of length less than *hashlen*×(2^{32}–1) bits. We assume that all key derivation function calls are indeed for bit strings of length less than *hashlen*×(2^{32}–1) bits. Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to *hashlen*×(2^{32}–1) bits shall output 'invalid' and stop. Similarly, it is assumed that all key derivation function calls do not involve hashing a bit string that is more than *maxhashlen* bits in length. Any scheme attempting to call the key derivation function on a call involving hashing a bit string that is greater than *maxhashlen* bits shall output "invalid" and stop.

C.5 The NIST SP 800-56A concatenation key derivation function

This clause describes a key derivation function based on the key derivation function that is given in the NIST Special Publication 800-56A.[32]

Function call: kdf (Z, OtherInput),

where OtherInput is *keydatalen* and OtherInfo.

Fixed Values (implementation dependent):

a) *hashlen*: an integer that indicates the length (in bits) of the output of the hash function used to derive blocks of secret keying material.

b) *max_hash_inputlen*: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

Auxiliary Function:

a) H: an approved hash function chosen from those specified in ISO/IEC 10118.

Input:

a) Z: a byte string that is the shared secret.

b) *keydatalen*: An integer that indicates the length (in bits) of the secret keying material to be generated; *keydatalen* shall be less than or equal to *hashlen* × (2^{32} –1).

c) OtherInfo: A bit string equal to the following concatenation:

AlgorithmID || EntityAInfo || EntityBInfo [|| SuppPubInfo] [|| SuppPrivInfo]

where the subfields are defined as follows:

a) AlgorithmID: A bit string that indicates how the derived keying material will be parsed and for which algorithm(s) the derived secret keying material will be used. For example, AlgorithmID might indicate that bits 1-80 are to be used as an 80-bit HMAC key and that bits 81-208 are to be used as a 128-bit AES key.

b) EntityAInfo: A bit string containing public information that is required by the application using this kdf to be contributed by entity *A* to the key derivation process. At a minimum, EntityAInfo shall include ID_A, the identifier of entity *A*. See the notes below.

c) EntityBInfo: A bit string containing public information that is required by the application using this kdf to be contributed by entity *B* to the key derivation process. At a minimum, EntityBInfo shall include ID_B, the identifier of entity *B*. See the notes below.

d) (Optional) SuppPubInfo: A bit string containing additional, mutually-known public information.

e) (Optional) SuppPrivInfo: A bit string containing additional, mutually-known private information (for example, a shared secret symmetric key that has been communicated through a separate channel).

Each of the three subfields AlgorithmID, EntityAInfo, and EntityBInfo shall be the concatenation of an application-specific, fixed-length sequence of substrings of information. Each substring representing a separate unit of information shall have one of these two formats: Either it is a fixed-length bit string, or it has the form Datalen || Data, where Data is a variable-length string of zero or more bytes, and Datalen is a fixed-length, big-endian counter that indicates the length (in bytes) of Data. (In this variable-length format, a null string of data shall be represented by using Datalen to indicate that Data has length zero.) An application using this kdf shall specify the ordering and number of the separate information substrings used in each of the subfields AlgorithmID, EntityAInfo, and EntityBInfo, and shall also specify which of the two formats (fixed-length or variable-length) is used for each substring. The application shall specify the lengths for all fixed-length quantities, including the Datalen counters.

The subfields SuppPrivInfo and SuppPubInfo (when allowed by the application) shall be formed by the concatenation of an application-specific, fixed-length sequence of substrings of additional information that may be used in key derivation upon mutual agreement of entities *A* and *B*. Each substring representing a separate unit of information shall be of the form Datalen || Data, where Data is a variable-length string of zero or more (eight-bit) bytes and Datalen is a fixed-length, big-endian counter that indicates the length (in bytes) of Data. The information substrings that entities *A* and *B* choose not to contribute are set equal to Null, and are represented in this variable-length format by setting Datalen equal to zero. If an application allows the use of the OtherInfo subfield SuppPrivInfo and/or the subfield SuppPubInfo, then the application shall specify the ordering and the number of additional information substrings that may be used in the allowed subfield(s) and shall specify the fixed-length of the Datalen counters.

Process:

a) reps = $\lceil keydatalen / hashlen \rceil$.

b) If reps > $(2^{32} - 1)$, then ABORT: output an error indicator and stop.

c) Initialize a 32-bit, big-endian bit string counter as 00000001 (in hexadecimal).

d) If counter || *Z* || OtherInfo is more than *max_hash_inputlen* bits long, then ABORT: output an error indicator and stop.

e) For *i* = 1 to reps by 1, do the following:

   1) Compute $Hash_i = H(\text{counter} \,||\, Z \,||\, \text{OtherInfo})$.

   2) Increment counter (modulo 2^{32}), treating it as an unsigned 32-bit integer.

f) Let Hhash be set to $Hash_{reps}$ if (*keydatalen* / *hashlen*) is an integer; otherwise, let Hhash be set to the (*keydatalen* mod *hashlen*) leftmost bits of $Hash_{reps}$.

g) Set DerivedKeyingMaterial = $Hash_1 \,||\, Hash_2 \,||\, \ldots \,||\, Hash_{reps-1} \,||\, Hhash$.

Output: The bit string DerivedKeyingMaterial of length *keydatalen* bits (or an error indicator). Any scheme attempting to call this key derivation function with *keydatalen* greater than or equal to *hashlen* × (2^{32} –1) shall output an error indicator and stop without outputting DerivedKeyingMaterial. Any call to the key derivation function involving an attempt to hash a bit string that is greater than *max_hash_inputlen* bits long shall cause the kdf to output an error indicator and stop without outputting DerivedKeyingMaterial.

NOTE 1 ID_A and ID_B shall be represented in OtherInfo as separate units of information, using either the fixed-length format or the variable-length format described above – according to the requirements of the application using this kdf.

NOTE 2 Entity *A* shall be the initiator, and entity *B* shall be the responder, as assigned by the protocol employing the key agreement scheme used to determine the shared secret *Z*.

C.6 The NIST SP 800-56A ASN.1 key derivation function

This clause describes a key derivation function based on the key derivation function that is given in the NIST Special Publication 800-56A.[32]

Function call: kdf (*Z*, OtherInput)

where OtherInput is *keydatalen* and OtherInfo.

Fixed Values (implementation dependent):

a) *hashlen*: an integer that indicates the length (in bits) of the output of the hash function used to derive blocks of secret keying material.

b) *max_hash_inputlen*: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

Auxiliary Function:

a) *H*: an approved hash function chosen from those specified in ISO/IEC 10118.

Input:

a) *Z*: a byte string that is the shared secret.

b) *keydatalen*: An integer that indicates the length (in bits) of the secret keying material to be generated; *keydatalen* shall be less than or equal to *hashlen* × (2^{32} –1).

c) OtherInfo: A bit string specified in ASN.1 DER encoding, which consists of the following information:

   1) AlgorithmID: A bit string that indicates how the derived keying material will be parsed and for which algorithm(s) the derived secret keying material will be used. For example, AlgorithmID might indicate that bits 1-80 are to be used as an 80-bit HMAC key and that bits 81-208 are to be used as a 128-bit AES key.

   2) EntityAInfo: A bit string containing public information that is required by the application using this kdf to be contributed by entity A to the key derivation process. At a minimum, EntityAInfo shall include ID_A, the identifier of entity *A*. See the notes below.

   3) EntityBInfo: A bit string containing public information that is required by the application using this kdf to be contributed by entity *B* to the key derivation process. At a minimum, EntityBInfo shall include ID_B, the identifier of entity *B*. See the notes below.

   4) (Optional) SuppPubInfo: A bit string containing additional, mutually-known public information.

   5) (Optional) SuppPrivInfo: A bit string containing additional, mutually-known private information (for example, a shared secret symmetric key that has been communicated through a separate channel).

Process:

a) reps = $\lceil keydatalen / hashlen \rceil$.

b) If reps > (2^{32} –1), then ABORT: output an error indicator and stop.

c) Initialize a 32-bit, big-endian bit string counter as 00000001 (in hexadecimal).

d) If counter || *Z* || OtherInfo is more than *max_hash_inputlen* bits long, then ABORT: output an error indicator and stop.

e) For *i* = 1 to reps by 1, do the following:

1) Compute $\text{Hash}_i = H(\text{counter} \,||\, Z \,||\, \text{OtherInfo})$.

2) Increment counter (modulo 2^{32}), treating it as an unsigned 32-bit integer.

f) Let Hhash be set to $\text{Hash}_{\text{reps}}$ if (*keydatalen* / *hashlen*) is an integer; otherwise, let Hhash be set to the (*keydatalen* mod *hashlen*) leftmost bits of $\text{Hash}_{\text{reps}}$.

g) Set DerivedKeyingMaterial = $\text{Hash}_1 \,||\, \text{Hash}_2 \,||\, \ldots \,||\, \text{Hash}_{\text{reps-1}} \,||\, \text{Hhash}$.

Output: The DerivedKeyingMaterial as a bit string of length *keydatalen* bits (or an error indicator). The ASN.1 kdf produces secret keying material that is at most *hashlen* × (2^{32}–1) bits in length. Any call to this key derivation function using a *keydatalen* value that is greater than *hashlen* × (2^{32}–1) shall cause the kdf to output an error indicator and stop without outputting DerivedKeyingMaterial. Any call to the key derivation function involving an attempt to hash a bit string that is greater than *max_hash_inputlen* bits long shall cause the kdf to output an error indicator and stop without outputting DerivedKeyingMaterial.

NOTE 1 ID_A and ID_B shall be represented in OtherInfo as separate units of information.

NOTE 2 Entity *A* shall be the initiator, and entity *B* shall be the responder, as assigned by the protocol employing the key agreement scheme used to determine the shared secret *Z*.

Annex D
(informative)

Examples of key establishment mechanisms

D.1 Examples of a function F, and sets S_1 and S_2

This annex first specifies a widely used example of a function F, and accompanying sets S_1 and S_2, which is conjectured to satisfy the five properties listed in Clause 10, given that certain parameters are chosen appropriately.

Let p be a prime number, and g be a primitive element of F_p. Let $S_2 = \{0, 1, ... p\text{-}1\}$, and $S_1 = \{2, ... , p - 2\}$. Then set $F(h, g) = g^h \bmod p$.

F is commutative with respect to h, where $(g^{h_B})^{h_A} = (g^{h_A})^{h_B} = (g^{h_A h_B}) \bmod p$.

The prime p shall be large enough so that F(·,g) can be conjectured to be a one-way function. Let each entity X have a private key h_x in S_1 which is only known by entity X, and a public key $p_X = g^{h_x} \bmod p$ known by all other entities.

NOTE 1 For discrete logarithm modulo a prime, the size of the prime should be chosen such that computing discrete logarithms in the corresponding cyclic group is computationally infeasible. Some other conditions on the prime number can be imposed in order to make discrete logarithms infeasible. It is also recommended to choose p to be a strong prime such that p - 1 has a large prime factor q and choose g to be a generator of a group of its large prime order q.

NOTE 2 For discrete logarithm modulo a composite, the modulus should be chosen as the product of two distinct odd primes that should be kept secret. The size of the modulus should be chosen such that factoring the modulus is computationally infeasible. Some additional conditions on the choice of the primes can be imposed in order to make factoring the modulus computationally infeasible.

D.2 Non-interactive Diffie-Hellman key agreement

This [20] is an example of key agreement mechanism 1.

Key Construction (A1) Entity A computes, using its own private key agreement key h_A and entity B's public key agreement key p_B, the shared key as $K_{AB} = p_B{}^{h_A} \bmod p$.

Key Construction (B1) Entity B computes, using its own private key agreement key h_B and entity A's public key agreement key p_A, the shared key as $K_{AB} = p_A{}^{h_B} \bmod p$.

D.3 Identity-based mechanism

This [23] is an example of key agreement mechanism 1, which is identity-based in the following sense:

— the public key of an entity can be retrieved from some combination of its identity and its certificate;

— the authenticity of the certificate is not directly verified, but the correct public key can only be recovered from an authentic certificate.

Let (n,y) be the public verification key of a certification authority, in the digital signature scheme giving message recovery which is specified in ISO/IEC 9796-2, Annex B (informative). Therefore n is the product of two large prime numbers p and q, kept secret by the certification authority, and y is co-prime with $lcm(p\text{-}1, q\text{-}1)$.

Let O be an integer of large order modulo n and $g = O^y \bmod n$.

Let I_X be the result of adding redundancy to a public information on entity X which contains at least the distinguishing identifier of entity X and possibly a serial number, a validity period, a time stamp and other data elements. Then entity X's key management pair is (h_X, p_X) where h_X is an integer less than n and $p_X = g^{h_X} \bmod n$. ISO/IEC 9796-3[4] is referred for a description of how to add redundancy.

Its certificate is computed by the certification authority as $Cert_X = s_X O^{h_X} \bmod n$, where s_X is the integer such that

$s_X{}^Y I_X = 1 \bmod n$.

Key Construction (A1) Entity A computes the public key of entity B as $p_B = Cert_B{}^Y \cdot I_B \bmod n$ and computes the shared secret key as $K_{AB} = p_B{}^{h_A} = g^{h_A h_B} \bmod n$.

Key Construction (B1) Entity B computes the public key of entity A as $p_A = Cert_A{}^Y \cdot I_A \bmod n$ and computes the shared secret key as $K_{AB} = p_A{}^{h_B} = g^{h_A h_B} \bmod n$.

NOTE A one-pass and a two-pass identity-based mechanisms using the same set-up are described in the references [23], [34] and [36] in the Bibliography.

D.4 ElGamal key agreement

This [21] is an example of key agreement mechanism 2.

One shall check that p to be a strong prime such that p - 1 has a large prime factor and that the exponentials are not of the form 0, +1, -1 mod p.

Key Token Construction (A1) Entity A randomly and secretly generates r in {2, ... , p-2 }, computes $g^r \bmod p$ and constructs the key token $KT_{A1} = g^r \bmod p$ and sends it to entity B.

Key Construction (A2) Entity A computes the shared key $K_{AB} = (p_B)^r \bmod p = g^{h_B r} \bmod p$.

Key Construction (B1) Entity B computes the shared key $K_{AB} = (g^r)^{h_B} \bmod p = g^{h_B r} \bmod p$.

D.5 Nyberg-Rueppel key agreement

This [33] is an example of key agreement mechanism 3. The signature system and the key agreement scheme are chosen in such a way that the signature system is determined by the keys (h_X, p_X).

Let q be a large prime divisor of p-1, g an element of F_p of order q, and set H = {2, ... , q-2}. Then entity X's asymmetric key pair used for signatures and key agreements is (h_X, p_X), where h_X is an element of H and

$p_X = g^{h_X} \bmod p$.

To prevent replay of old key tokens this example makes use of a time-stamp or a serial number, TVP, and of a cryptographic hash function hash, which maps strings of bits of arbitrary length to random integers in a large subset of {2, ... , p-2}, for example, in H.

NOTE A hash-function as defined here is collision resistant.

Key Construction (A1.1) Entity A randomly and secretly generates r in H and computes $e = g^r \bmod p$.

Further entity A computes the shared secret key as $K_{AB} = p_B{}^r \bmod p$.

Using the shared secret key K_{AB}, entity A computes a MAC on the sender's distinguishing identifier for entity A and a sequence number or time-stamp TVP, $e' = e \cdot \text{hash}(K_{AB}||A||TVP) \bmod p$.

Key Token Signature (A1.2) Entity A computes the signature $y = r - h_A e' \bmod q$.

Entity A forms the key token $KT_{A1} = A||e||TVP||y$ and sends it to entity B.

Key Construction (B1.1) Entity B computes the shared secret key, using its private key agreement key h_B,

$K_{AB} = e^{h_B} \bmod p$.

Using the shared secret key K_{AB}, entity B computes the MAC on the sender's distinguishing identifier for entity A and the TVP, and computes $e' = e \cdot \mathrm{hash}(K_{AB}||A||TVP) \bmod p$.

Signature Verification (B1.2) Entity B checks the validity of TVP and verifies, using the sender's public key p_A, the equality $e = g^y p_A{}^{e'} \bmod p$.

D.6 Diffie-Hellman key agreement

This [20] is an example of key agreement mechanism 4.

One shall check that p to be a strong prime such that p - 1 has a large prime factor and that the exponentials are not of the form 0, +1, -1 mod p.

Key Token Construction (A1) Entity A randomly and secretly generates r_A in {2, ... , p-2 }, computes g^{r_A} mod p, constructs the key token as $KT_{A1} = g^{r_A} \bmod p$, and sends it to entity B.

Key Token Construction (B1) Entity B randomly and secretly generates r_B in {2, ... , p-2 }, computes g^{r_B} mod p, constructs the key token, $KT_{B1} = g^{r_B} \bmod p$, and sends it to entity A.

Key Construction (A2) Entity A computes the shared key as $KT_{AB} = (g^{r_B})^{r_A} = g^{r_A r_B} \bmod p$.

Key Construction (B2) Entity B computes the shared key as $K_{AB} = (g^{r_A})^{r_B} = g^{r_A r_B} \bmod p$.

D.7 Matsumoto-Takashima-Imai A(0) key agreement

This [28] is an example of key agreement mechanism 5.

One recommended method is to use a safe prime p and to check that the exponentials are not of the form 0, +1, -1 mod p.

Key Token Construction (A1) Entity A randomly and secretly generates r_A in {2, ... , p-2}, computes the key token as $KT_{A1} = g^{r_A} \bmod p$ and sends it to entity B.

Key Token Construction (B1) Entity B randomly and secretly generates r_B in {2, ... , p-2}, computes the key token as $KT_{B1} = g^{r_B} \bmod p$ and sends it to entity A.

Key Construction (B2) Entity B computes the shared key as $K_{AB} = \mathrm{w}(KT_{A1}{}^{h_B}, p_A{}^{r_B}) = KT_{A1}{}^{h_B}\, p_A{}^{r_B} \bmod p$.

Key Construction (A2) Entity A computes the shared key as $K_{AB} = \mathrm{w}(p_B{}^{r_A}, KT_{B1}{}^{h_A}) = KT_{A1}{}^{h_B}\, p_A{}^{r_B} \bmod p$.

NOTE To avoid attacks in,[25] each entity needs to reject a trivial value of KTA1 or KTB1 = 0 or 1 and the same private keys $h_A = h_B$.

D.8 Beller-Yacobi protocol

This clause gives a description of the original Beller-Yacobi protocol,[17] which has been used to derive key agreement mechanism 6.

NOTE This mechanism is not completely compatible with the Mechanism 6 as it was optimized for specific situations. Specifically it uses ElGamal signature scheme and makes use of an additional symmetric encryption algorithm to transfer entity B's signature verification key and its certificate to entity A in a confidential way, thus assuring anonymity.

Let enc: $K : M \rightarrow C$ be a conventional encryption function, such as the algorithms found in ISO/IEC 18033-3, where K = key space, M = message space, and C = cryptogram space.

Let S_X denote the ElGamal signature operation of entity X. The process described below emphasizes the distinction between off-line and on-line operations required in ElGamal family of signature schemes.

We use P_X and C_X to denote entity X 's public key and certificate, respectively. The public encryption operation of entity X (which uses P_X) is denoted E_X (modular squaring in the case of Rabin).

Off-line computation: entity B picks a random number r_B and computes $u = g^{r_B} \bmod p$.

Key Token Construction (A1) Entity A picks a random number r_A and computes $KT_{A1} = (r_A||A||C_A)$ and sends it to entity B.

Key Token Processing (B1) Entity B produces the signature $BS = (u,v) = S_B(r_A||A)$, where u and v is the ElGamal signature. Then entity B picks a random x_B and creates $KT_{B1} = E_A(BS)||enc(u,(B||P_B||C_B||x_B))$ and sends it to entity A.

Key Construction (B2) The shared secret key consists of part of entity B's signature, u.

Key Token Processing and Key Construction (A2) Entity A decrypts the key token $E_A(BS)$ to find the session key u, then decrypts the conventional encryption $enc(u,(B||P_B||C_B||x_B))$ using session key u to find the identifier, public key, and certificate of the alleged entity B. Entity A verifies certificate C_B, and if positive it then uses the verification function, V_B to verify entity B's signature BS. If positive it then accepts u as a shared secret key.

Annex E
(informative)

Examples of elliptic curve based key establishment mechanisms

E.1 Example of a function F

This annex first gives a widely used example of a function F to satisfy the five properties listed in Clause 10, given that certain parameters are chosen appropriately.

Let E be an elliptic curve defined over a finite field F_q. Given an integer d and a point G in $E(\mathrm{GF}(q))$ where G is the base point, then the function F is $\mathrm{F}(d,G) = dG$.

F has the property that $d_1(d_2G) = d_2(d_1G) = d_1d_2G$.

$E(F_q)$ shall be large enough so that $\mathrm{F}(\cdot,G)$ can be conjectured to be a one-way function. Let each entity X have a private key h_X in $E(\mathrm{F}(q))$, which is only known by entity X, and a public key $p_X = h_X G$ known by all other entities.

E.2 Common information

For all key agreement mechanisms, prior to the process of agreeing upon a shared secret, the following common information shall be established between the parties and optionally validated (ISO/IEC 15946-1 is referred for a description of parameter validation):

— the elliptic curve parameters with which the key pairs shall be associated, which shall be the same for both parties key pairs. This includes p, p^m, 2^m, or 3^m, a description of GF(q), GF(p^m) ,GF(2^m), or GF(3^m) and an indication of the basis used, E, n and G.

Named curve identifiers such as those specified in X9.62, provide a simple means of identifying elliptic curve domain parameters and can be used to specify groups of common information values.

In each of the mechanisms defined below, the resulting agreed key should not be used as a cryptographic key directly. Instead, it should be used as the input to a key derivation function, allowing both parties to derive the same cryptographic keys from it. Hence, it is also necessary for the two parties to agree on the following information:

— a key derivation function, kdf;

— any parameters to the key derivation function, and

— the type of cofactor multiplication that is to be performed (if any).

E.3 Non-interactive key agreement of Diffie-Hellman type

This [20] is an example of key agreement mechanism 1. This key agreement mechanism non-interactively establishes a shared secret between two entities A and B.

Prior to the process of agreeing upon a shared secret, in addition to the common information, the following shall be established:

— for each entity X, a private key-agreement key h_X and a public key-agreement key P_X, which is an elliptic curve point satisfying $P_X = h_XG$. ISO/IEC 15946-1 is referred for a description of how to generate this key pair.

— for each entity, access to an authentic copy of the public key-agreement key of the other party.

Each entity shall independently verify that the other entity's public key is indeed a point on the elliptic curve. ISO/IEC 15946-1 is referred for a description of how to do this.

The values l and j are used for cofactor multiplication as explained in Clause 7.

Key construction (A1) Entity A computes, using its own private key-agreement key h_A and entity B's public key-agreement key P_B, the shared key as $K_{AB} = (h_A \cdot l)(j \cdot P_B)$.

Key construction (B1) Entity B computes, using its own private key-agreement key h_B and entity A's public key-agreement key P_A, the shared key as $K_{AB} = (h_B \cdot l)(j \cdot P_A)$.

NOTE As a consequence of the first property, the established secret between the same two users always has the same value. For this reason it is suggested that the input to the key derivation function in this case include time-varying information.

E.4 Key agreement of ElGamal type

This [21] is an example of key agreement mechanism 2. This key agreement mechanism establishes a shared secret between two entities A and B in one pass.

Prior to the process of agreeing upon a shared secret, in addition to the common information, the following shall be established:

— for entity B, a private key-agreement key d_B and a public key-agreement key P_B, which is an elliptic curve point satisfying $P_B = d_B G$. ISO/IEC 15946-1 is referred for a description of how to generate this key pair.

— for entity A, access to an authentic copy of the public key-agreement key of entity B.

Entity A should verify that entity B's public key is indeed a point on the elliptic curve. ISO/IEC 15946-1 is referred for a description of how to do this.

The values l and j are used for cofactor multiplication as explained in Clause 7.

Key token construction (A1.1) Entity A randomly and secretly generates r in the range $\{2,...,n\text{-}2\}$, computes rG, constructs the key token, $KT_{A1} = rG$, and sends it to entity B.

Key construction (A1.2) Entity A computes the shared key as $K_{AB} = (r \cdot l)(j \cdot P_B)$.

Key construction (B1) Entity B should verify that KT_{A1} is indeed a point on the elliptic curve. A description of how to do this is referred in ISO/IEC 15946-1. Using its own private key, entity B computes the shared key from KT_{A1} as follows: $K_{AB} = (d_B \cdot l)(j \cdot KT_{A1})$.

NOTE This key agreement mechanism provides forward secrecy with respect to entity A.

E.5 Key agreement following Nyberg-Rueppel

This [33] is an example of key agreement mechanism 3. The protocol is not a 1-1-transcript of protocol C.4; but follows the essential ideas of C.4.

The signature system and the key agreement scheme are chosen in such a way that the signature system is determined by the keys (h_X, P_X).

Let q be a large prime divisor of p-1, g an element of F_p of order q, and set $H = \{2, \ldots, q\text{-}2\}$. Then entity X's asymmetric key pair used for signatures and key agreements is (h_X, p_X), where h_X is an element of H and

$p_X = g^{h_X} \bmod p.$

To prevent the replay of old key tokens this example makes use of a timestamp or a serial number TVP, and of a cryptographic hash function hash, which maps strings of bits of arbitrary length to random integers into H, for example.

The values l and j are used for cofactor multiplication as explained in Clause 7.

NOTE A hash-function as defined here is collision resistant.

Key Construction (A1.1) Entity A randomly and secretly generates r in H and computes $R = rG$.

Further entity A computes the shared secret key as $K_{AB} = (r \cdot l)(j \cdot P_B)$.

Using the shared secret key K_{AB}, entity A computes a MAC on the point R, the sender's distinguishing identifier for entity A and a sequence number or timestamp TVP: e =hash(R||K_{AB}||A||TVP).

Key Token Signature (A1.2) Entity A computes the signature $y = (r\text{-}h_A e) \bmod q$, forms the key token

KT_{A1} = (R||A||TVP||y) and sends it to entity B.

Key Construction (B1.1) Entity B computes the shared secret key, using its private key agreement key h_B,

$K_{AB} = (h_B \cdot l)(j \cdot R)$.

Using the shared secret key K_{AB} entity B computes the MAC on the sender's distinguishing identifier for entity A and the TVP and computes e =hash(R||K_{AB}||A||TVP).

Signature Verification (B1.2) Entity B checks the validity of TVP and verifies, using the sender's public key P_A, the equality $R = yG + eP_A$.

E.6 Key agreement of Matsumoto-Takashima-Imai type A(0)

This [28] is an example of key agreement mechanism 5.

Let q be a large prime divisor of p-1, g an element of F_p of order q, and set $H = \{2, \ldots, q\text{-}2\}$.

The values l and j are used for cofactor multiplication as explained in Clause 7.

Key Token Construction (A1) Entity A randomly and secretly generates r_A in H, computes the key token

$KT_{A1} = (r_A \cdot l)(j \cdot G)$, and sends it to entity B.

Key Token Construction (B1) Entity B randomly and secretly generates r_B in H, computes the key token

$KT_{B1} = (r_B \cdot l)(j \cdot G)$, and sends it to entity A.

Key Construction (B2) Entity B computes the shared key as $K_{AB} = \mathrm{w}(h_B KT_{A1}, r_B P_A)$, where w is a one-way function.

Key Construction (A2) Entity A computes the shared key as $K_{AB} = \mathrm{w}(h_A KT_{B1}, r_A P_B)$.

E.7 Key agreement of Diffie-Hellman type

This[20] is an example of key agreement mechanism 4. This key agreement mechanism establishes a shared secret between entities A and B in two passes.

This key agreement mechanism does not require any initial information other than the common information to be set up. The values l and j are used for cofactor multiplication as explained in Clause 7.

Key Token Construction (A1) Entity A randomly and secretly generates r_A in the range $\{2,...,n\text{-}2\}$, computes r_AG, constructs the key token, $KT_{A1} = r_AG$, and sends it to entity B.

Key Token Construction (B1) Entity B randomly and secretly generates r_B in the range $\{2,...,n\text{-}2\}$, computes r_BG, constructs the key token, $KT_{B1} = r_BG$, and sends it to entity A.

Key Construction (A2) Entity A should verify that KT_{B1} is indeed a point on the elliptic curve. A description of how to do this is referred in ISO/IEC 15946-1. Entity A computes the shared key $K_{AB} = (r_A{\cdot}l)(j{\cdot}KT_{B1})$.

Key Construction (B2) Entity B should verify that KT_{A1} is indeed a point on the elliptic curve. A description of how to do this is referred in ISO/IEC 15946-1. Entity B computes the shared key $K_{AB} = (r_B{\cdot}l)(j{\cdot}KT_{A1})$.

NOTE This key agreement mechanism provides mutual forward secrecy.

E.8 Key agreement of Diffie-Hellman type with 2 key pairs

This key agreement mechanism establishes a shared secret between entities A and B in two passes.

Prior to the process of agreeing upon a shared secret, in addition to the common information, the following shall be established:

— for each entity X, a private key-agreement key d_X and a public key-agreement key P_X, which is an elliptic curve point satisfying $P_X = d_XG$. ISO/IEC 15946-1 is referred for a description of how to generate this key pair.

— for each entity, access to an authentic copy of the public key-agreement key of the other party.

Each entity should independently verify that the other entity's public key is indeed a point on the elliptic curve. ISO/IEC 15946-1 is referred for a description of how to do this.

The values l and j are used for cofactor multiplication as explained in Clause 7.

Key token construction (A1) Entity A randomly and secretly generates r_A in the range $\{2,...,n\text{-}2\}$, computes r_AG, constructs the key token, $KT_{A1} = r_AG$, and sends it to entity B.

Key token construction (B1) Entity B randomly and secretly generates r_B in the range $\{2,...,n\text{-}2\}$, computes r_BG, constructs the key token, $KT_{B1} = r_BG$, and sends it to entity A.

Key construction (A2): Entity A should verify that KT_{B1} is indeed a point on the elliptic curve. A description of how to do this is referred in ISO/IEC 15946-1. Entity A computes the shared key $K_{AB} = (d_A{\cdot}l)(j{\cdot}KT_{B1})||(r_A{\cdot}l)(j{\cdot}P_B)$.

Key construction (B2) Entity B should verify that KT_{A1} is indeed a point on the elliptic curve. A description of how to do this is referred in ISO/IEC 15946-1. Entity B computes the shared secret $K_{AB} = (r_B{\cdot}l)(j{\cdot}P_A)||(d_B{\cdot}l)(j{\cdot}KT_{A1})$.

NOTE 1 Concatenation of a representation of the points is not the only alternative for the construction of the key. Any prefix free representation (such as ASN.1) will also work. As there are choices, the method to combine the two values becomes part of what is needed to be agreed upon by all parties. See also Annex C for further discussion.

NOTE 2 The number of passes is 2.

NOTE 3 This mechanism provides forward secrecy with respect to both entity A and B individually.

NOTE 4 This mechanism provides mutual implicit key authentication.

E.9 Key agreement of Diffie-Hellman type with 2 signatures and key confirmation

This is an example of key agreement mechanism 7.

This key agreement mechanism establishes a shared secret between entities *A* and *B* in three passes.

Prior to the process of agreeing upon a shared secret, in addition to the common information, the following shall be established:

— for each entity, a private signature key and a public verification key corresponding to a mutually agreed upon signature algorithm.

— for each entity, access to an authentic copy of the public verification key of the other party.

— any parameters to be used in the signature transformations.

— a MAC function.

Let *X* 's private and public signature transformations be denoted S_X and V_X respectively; (S_X, V_X) could denote any signature system. Both ISO/IEC 9796-2,[3] ISO/IEC 9796-3,[4] and ISO/IEC 14888[9] are referred for signature systems.

The values *l* and *j* are used for cofactor multiplication as explained in Clause 7.

Key token construction (A1) Entity *A* randomly and secretly generates r_A in the range {2,...,*n*-2}, computes r_AG, constructs the key token $KT_{A1} = r_AG$, and sends it to entity *B*.

Key token processing and key construction (B1) Entity *B* should verify that KT_{A1} is indeed a point on the elliptic curve. ISO/IEC 15946-1 is referred for a description of how to do this. Entity *B* randomly and secretly generates r_B in the range {2,...,*n*-2}, computes r_BG, computes the shared secret as $K_{AB} = (r_B{\cdot}l)(j{\cdot}KT_{A1})$, constructs the signed key token, $KT_{B1} = S_B(DB_1)||MAC_{K_{AB}}(DB_1)$ for $DB_1 = r_BG||KT_{A1}||A||Text1$, and sends it to entity *A*.

NOTE 1 As a way to reduce the amount of data transmitted, if a signature scheme with appendix is used, the redundant value KT_{A1} need not be returned with the block KT_{B1}, although it still shall be included within the scope of the signature calculation.

Key token processing (A2) Entity A verifies B's signature on the key token KT_{B1} using entity B's public verification key. If a signature scheme with message recovery is used, this includes recovering the data block DB_1 from the signature and verifying that entity A's distinguishing identifier and the value r_AG are contained in it. If a signature scheme with appendix is used, this includes reconstructing the data block DB_1 using the value in KT_{A1}, entity A's distinguishing identifier and the received value r_BG and verifying the signature on that data block.

Entity *A* should verify that the value r_BG obtained from KT_{B1} is indeed a point on the elliptic curve. ISO/IEC 15946-1 is referred for a description of how to do this. If the checks are successful, entity *A* computes the shared key $K_{AB} = (r_A{\cdot}l)(j{\cdot}r_BG)$.

Using K_{AB}, entity *A* verifies $MAC_{K_{AB}}(DB_1)$. Then entity *A* constructs the signed key token

$KT_{A2} = S_A(DB_2)||MAC_{K_{AB}}(DB_2)$, where $DB_2 = r_AG||r_BG||B||Text2$ and sends it to entity *B*.

NOTE 2 As a way to reduce the amount of data transmitted, if a signature scheme with appendix is used, the redundant values r_AG and r_BG need not be returned with the block KT_{A2}, although they still shall be included within the scope of the signature calculation.

Key token processing (B2) Entity *B* verifies entity A's signature on the key token KT_{A2} using entity A's public verification key. If a signature scheme with message recovery is used, this includes recovering the data block DB_2 from the signature and verifying that entity B's distinguishing identifier and the values r_AG and r_BG are contained in it. If a signature scheme with appendix is used, this includes

