Draft NIST Special Publication 800-56B

December, 2008

Recommendation for Pair-Wise Key Establishment Schemes: Using Integer Factorization Cryptography

Elaine Barker, Lily Chen, Andrew Regenscheid, and Miles Smid

C O M P U T E R S E C U R I T Y

Abstract

This Recommendation specifies key establishment schemes using *integer* factorization cryptography, based on ANS X9.44, *Key Establishment using Integer Factorization Cryptography,* which was developed by the Accredited Standards Committee (ASC) X9, Inc.

KEY WORDS: assurances; integer factorization cryptography; key agreement; key confirmation; key derivation; key establishment; key management; key recovery; key transport.

Acknowledgements

The National Institute of Standards and Technology (NIST) gratefully acknowledges and appreciates contributions by Rich Davis from the National Security Agency concerning the many security issues associated with this Recommendation. NIST also thanks the many contributions by the public and private sectors whose thoughtful and constructive comments improved the quality and usefulness of this publication.

Authority

This document has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347.

NIST is responsible for developing standards and guidelines, including minimum requirements, for providing adequate information security for all agency operations and assets, but such standards and guidelines shall not apply to national security systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130, Section 8b(3), Securing Agency Information Systems, as analyzed in A-130, Appendix IV: Analysis of Key Sections. Supplemental information is provided in A-130, Appendix III.

This Recommendation has been prepared for use by federal agencies. It may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright. (Attribution would be appreciated by NIST.)

Nothing in this document should be taken to contradict standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official.

Conformance testing for implementations of key establishment schemes, as specified in this Recommendation, will be conducted within the framework of the Cryptographic Module Validation Program (CMVP), a joint effort of NIST and the Communications Security Establishment of the Government of Canada. An implementation of a key establishment scheme must adhere to the requirements in this Recommendation in order to be validated under the CMVP. The requirements of this Recommendation are indicated by the word “shall.”

Table of Contents

1 Introduction 10
2 Scope and Purpose 10
3 Definitions, Symbols and Abbreviations 11
3.1 Definitions 11
3.2 Symbols and Abbreviations 17
4 Key Establishment Schemes Overview 23
4.1 Key Establishment Preparations by an Owner 24
4.2 Key Agreement Process 25
4.3 IFC-based Key Transport Process 28
5 Cryptographic Elements 30
5.1 Cryptographic Hash Functions 30
5.2 Message Authentication Code (MAC) Algorithm 30
5.2.1 *MacTag* Computation 30
5.2.2 *MacTag* Checking 31
5.2.3 Implementation Validation Message 31
5.3 Random Bit Generation 31
5.4 Prime Number Generators 31
5.5 Primality Testing Methods 32
5.6 Nonces 32
5.7 Symmetric Key-Wrapping Algorithms 32
5.8 Mask Generation Function (MGF) 33
5.9 Key Derivation Functions for Key Establishment Schemes 34
5.9.1 Concatenation Key Derivation Function (Approved Alternative 1) 34
5.9.2 ASN.1 Key Derivation Function (Approved Alternative 2) 37
6 RSA Key Pairs 39

6.1 General Requirements 39
6.2 Criteria for RSA Key Pairs for Key Establishment 40
6.2.1 Definition of a Key Pair 40
6.2.2 Formats 42
6.2.3 Parameter Length Sets 42
6.3 RSA Key Pair Generators 43
6.3.1 RSAKPG1 Family: RSA Key Pair Generation with a Fixed Public Exponent 43
6.3.2 RSAKPG2 Family: RSA key pair generation with a random public exponent 46
6.4 Assurances of Validity 49
6.4.1 Assurance of Key Pair Validity 49
6.4.2 Recipient Assurances of Public Key Validity 55
6.5 Assurances of Private Key Possession 56
6.5.1 Owner Assurance of Private Key Possession 57
6.5.2 Recipient Assurance of Owner's Possession of a Private Key 57
6.6 Key Confirmation 59
6.6.1 Unilateral Key Confirmation for Key Establishment Schemes 61
6.6.2 Bilateral Key Confirmation for Key Establishment Schemes 62
6.7 Authentication 62
7 IFC Primitives and Operations 63
7.1 Encryption and Decryption Primitives 63
7.1.1 RSAEP 63
7.1.2 RSADP 63
7.2 Encryption and Decryption Operations 64
7.2.1 RSA Secret Value Encapsulation (RSASVE) 64
7.2.2 RSA with Optimal Asymmetric Encryption Padding (RSA-OAEP) 67
7.2.3 RSA-based Key-Encapsulation Mechanism with a Key-Wrapping Scheme (RSA-KEM-KWS) 74
8 Key Agreement Schemes 79

8.1 Common Components for Key Agreement 80
8.2 The KAS1 Family 80
8.2.1 KAS1 Family Prerequisites 81
8.2.2 KAS1-basic 81
8.2.3 KAS1 Key Confirmation 83
8.2.4 KAS1 Security Properties 85
8.3 KAS2 Family 86
8.3.1 KAS2 Family Prerequisites 87
8.3.2 KAS2-basic 88
8.3.3 KAS2 Key Confirmation 90
8.3.4 KAS2 Security Properties 95
9 IFC based Key Transport Schemes 96
9.1 Additional Input 97
9.2 KTS-OAEP Family: Key Transport Using RSA-OAEP 97
9.2.1 KTS-OAEP Family Prerequisites 98
9.2.2 Common components 99
9.2.3 KTS-OAEP-basic 99
9.2.4 KTS-OAEP Key Confirmation 100
9.2.5 KTS-OAEP Security Properties 101
9.3 KTS-KEM-KWS Family: Key Transport using RSA-KEM-KWS 102
9.3.1 KTS-KEM-KWS Family Prerequisites 103
9.3.2 Common Components of the KTS-KEM-KWS Schemes 104
9.3.3 KTS-KEM-KWS-basic 104
9.3.4 KTS-KEM-KWS Key Confirmation 105
9.3.5 KTS-KEM-KWS Security Properties 106
10 Key Recovery 107
11. Implementation Validation 108

Appendix A: Summary of Differences between this Recommendation and ANS X9.44 (Informative) 110

Appendix B: Data Conversions (Normative) 112

B.1 Integer-to-Byte String (I2BS) Conversion 112

B.2 Byte String to Integer (BS2I) Conversion 112

Appendix C: Prime Factor Recovery (Normative) 113

Appendix D: References (Informative) 115

Figures

Figure 1: Owner Key Establishment Preparations 25
Figure 2: Key Agreement Process 26
Figure 3: Key Transport Process 28
Figure 4: RSA-OAEP Encryption Operation 70
Figure 5: RSA-OAEP Decryption Operation 73
Figure 6: RSA-KEM-KWS Encryption Operation 76
Figure 7: RSA-KEM-KWS Decryption Operation 79
Figure 8: KAS1-basic Scheme 83
Figure 9: KAS1-responder-confirmation Scheme (from Party V to Party U) 85
Figure 10: KAS2-basic Scheme 90
Figure 11: KAS2-responder-confirmation Scheme (from Party V to Party U) 92
Figure 12: KAS2-initiator-confirmation Scheme (from Party U to Party V) 93
Figure 13: KAS2-bilateral-confirmation Scheme 95
Figure 14: KTS-OAEP-basic Scheme 100
Figure 15: KET-OAEP-receiver-confirmation Scheme 101
Figure 16: KTS-KEM-KES-basic Scheme 105
Figure 17: KTS-KEM-KWS-receiver-confirmation Scheme 106

Tables

Table 1: IFC Parameters for Key Establishment 43

1 Introduction

Many U.S. Government Information Technology (IT) systems need to employ strong cryptographic schemes to protect the integrity and confidentiality of the data that they process. Algorithms such as the Advanced Encryption Standard (AES) as defined in Federal Information Processing Standard (FIPS) 197, Triple DES as specified in NIST Special Publication (SP) 800-67, and HMAC as defined in FIPS 198-1 [5] make attractive choices for the provision of these services. These algorithms have been standardized to facilitate interoperability between systems. However, the use of these algorithms requires the establishment of shared secret keying material in advance. Trusted couriers may manually distribute this secret keying material, but as the number of entities using a system grows, the work involved in the distribution of the secret keying material grows rapidly. Therefore, it is essential to support the cryptographic algorithms used in modern U.S. Government applications with automated key establishment schemes.

2 Scope and Purpose

This Recommendation provides the specifications of key establishment schemes that are appropriate for use by the U.S. Federal Government, based on a standard developed by the Accredited Standards Committee (ASC) X9, Inc.: ANS X9.44, *Key Establishment using Integer Factorization Cryptography* [10]. A key establishment scheme can be characterized as either a key agreement scheme or a key transport scheme. This Recommendation provides asymmetric-based key agreement and key transport schemes that are based on the Rivest Shamir Adleman (RSA) algorithm.

When there are differences between this Recommendation and the referenced ANS X9.44[10] standard, this key establishment schemes Recommendation **shall** have precedence for U.S. Government applications.

This Recommendation is intended for use in conjunction with NIST Special Publication 800-57, *Recommendation for Key Management* [7]. This key establishment schemes Recommendation, the Recommendation for Key Management [7], and FIPS 186-3 [3] standard are intended to provide information for a vendor to implement secure key establishment using asymmetric algorithms in FIPS 140-2/3 [1] validated modules.

A scheme may be a component of a larger protocol, which in turn provides additional security properties not provided by the scheme when considered by itself. Note that protocols, per se, are not specified in this Recommendation.

3 Definitions, Symbols and Abbreviations

3.1 Definitions

Additional input	Information known by two parties that is cryptographically bound to keying material using the encryption scheme.
Algorithm	A clearly specified mathematical process for computation; a set of rules which, if followed, will give a prescribed result.
Algorithm identifier	A unique identifier for a given cryptographic algorithm, together with any required parameters.
Approved	FIPS-**approved** or NIST-**recommended**. An algorithm or technique that meets at least one of the following: 1) is specified in a FIPS or NIST Recommendation, 2) is adopted in a FIPS or NIST Recommendation or 3) is specified in a list of NIST-**approved** security functions (e.g., specified as **approved** in the annexes of FIPS 140-2/3).
Assurance of possession of a private key	Confidence that an entity possesses a private key associated with a given public key.
Assurance of validity	Confidence that either a key or a set of domain parameters is arithmetically correct.
Bit length	The length in bits of a bit string.
Bit string	An ordered sequence of 0's and 1's.
Byte	A bit string of length 8. A byte is represented by a hexadecimal string of length 2. The right-most hexadecimal character represents the right-most four bits of the byte, and the left-most four bits of the byte represent the left-most four bits of the byte. For example, 9d represents the bit string 10011101.
Byte string	A sequence of bytes.
Certification Authority (CA)	The entity in a Public Key Infrastructure (PKI) that is responsible for issuing public key certificates and exacting compliance to a PKI policy.
Ciphertext	Data in its enciphered form.

Cryptographic key (Key)	A parameter used with a cryptographic algorithm that determines its operation. Examples include: 1. the transformation of plaintext data into ciphertext data, 2. the transformation of ciphertext data into plaintext data, 3. the computation of a digital signature from data, 4. the verification of a digital signature, 5. the computation of an authentication code from data, 6. the verification of an authentication code from data and a received authentication code, and 7. the computation of a shared secret that is used to derive keying material.
Data integrity	A property whereby data has not been altered in an unauthorized manner since it was created, transmitted or stored. In this Recommendation, the statement that a cryptographic algorithm "provides data integrity" means that the algorithm is used to detect unauthorized alterations.
Decryption	The process of transforming ciphertext into plaintext using a cryptographic algorithm and key.
Digital signature	The result of a cryptographic transformation of data that, when properly implemented with supporting infrastructure and policy, provides the services of: 1. origin authentication, 2. data integrity, and 3. signer non-repudiation.
Encryption	The process of transforming plaintext into ciphertext using a cryptographic algorithm and key.
Entity	An individual (person), organization, device, or process. “Party” is a synonym.
Entity authentication	A process that establishes the origin of information, or determines an entity’s identity to the extent permitted by the entity’s identifier.

Hash function	A function that maps a bit string of arbitrary length to a fixed length bit string. **Approved** hash functions are designed to satisfy the following properties: 1. (One-way) It is computationally infeasible to find any input that maps to any pre-specified output, and 2. (Collision resistant) It is computationally infeasible to find any two distinct inputs that map to the same output. **Approved** hash functions are specified in FIPS 180-3 [2].
Hash of a bit string	The hash value produced by applying a hash function to the bit string.
Hash value	The fixed-length bit string produced by a hash function.
Identifier	A bit string that is associated with a person, device or organization. It may be an identifying name, or may be something more abstract (for example, a string consisting of an Internet Protocol (IP) address and timestamp). If a party owns a key pair that is used in a key establishment transaction, then the identifier assigned to that party is one that is cryptographically bound to that key pair. If the party's key pair is not used in a key establishment transaction, then the identifier of that party is a non-null identifier selected in accordance with the protocol utilizing the scheme.
Initiator	The party that begins a key agreement transaction. Contrast with responder.
Key agreement	A key establishment procedure where the resultant secret keying material is a function of information contributed by two participants, so that no party can predetermine the value of the secret keying material independently from the contributions of the other party. Contrast with key transport.
Key agreement transaction	The instance that results in shared secret keying material among different parties using a key agreement scheme.
Key confirmation	A procedure to provide assurance to one party (the key confirmation recipient) that another party (the key confirmation provider) actually possesses the correct secret keying material and/or shared secret.
Key derivation	In this Recommendation, the process by which keying material is derived from a shared secret and other information.

Key establishment	The procedure that results in shared secret keying material among different parties.
Key establishment transaction	An instance of establishing secret keying material using a key establishment scheme.
Key management	The activities involving the handling of cryptographic keys and other related security parameters (e.g., IVs and passwords) during the entire life cycle of the keys, including their generation, storage, establishment, entry and output, and destruction.
Key pair	A public key and its corresponding private key; a key pair is used with a public key algorithm.
Key transport	A key establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement.
Key transport transaction	The instance that results in shared secret keying material between different parties using a key transport scheme.
Key wrap	A method of encrypting keying material (along with associated integrity information) that provides both confidentiality and integrity protection using a symmetric key algorithm.
Keying material	The data that is necessary to establish and maintain a cryptographic keying relationship. Some keying material may be secret, while other keying material may be public. As used in this Recommendation, secret keying material may include keys, secret initialization vectors or other secret information; public keying material includes any non-secret data needed to establish a relationship.
Length in bits of an integer, x	The length, in bits, of the shortest bit string containing the binary representation of x. For example, the length in bits of 5 is 3.
Length in bytes of an integer, x	The length, in bytes, of the shortest byte string containing the binary representation of x. For example, the length in bytes of 5 is 1.
Message Authentication Code (MAC) algorithm	A family of one-way cryptographic functions that is parameterized by a symmetric key. A given function in the family produces a *MacTag* on input data of arbitrary length. A MAC algorithm can be used to provide data origin authentication as well as data integrity. In this Recommendation, a MAC algorithm is used for key confirmation and validation testing purposes.

Nonce	A time-varying value that has at most a negligible chance of repeating. For example, a nonce is a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.
Owner	For a key pair, the owner is the entity that is authorized to use the private key associated with a public key, whether that entity generated the key pair itself or a trusted party generated the key pair for the entity.
Party	An individual (person), organization, device, or process. "Entity" is a synonym for party.
Prime number	An integer that is greater than 1 and divisible only by 1 and itself.
Primitive	A low level cryptographic algorithm used as a basic building block for higher level cryptographic operations or schemes.
Private key	A cryptographic key, used with a public key cryptographic algorithm that is kept secret. A private key is associated with a public key.
Protocol	A special set of rules used by two or more entities that describe the message order and data structures for information exchanged between the entities.
Provider	A party that provides (1) a public key (e.g., in a certificate); (2) assurance, such as an assurance of the validity of a candidate public key or assurance of possession of the private key associated with a public key; or (3) key confirmation. Contrast with recipient.
Public key	A cryptographic key, used with a public key cryptographic algorithm that may be made public. A public key is associated with a private key.
Public key algorithm	A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that determining the private key from the public key is computationally infeasible.
Public key certificate (certificate)	A set of data that uniquely identifies an entity's identifiers, the entity's public key, and possibly other information, and is digitally signed by a trusted party, thereby binding the public key to the included identifier(s).

Public key cryptography	A form of cryptography that uses two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key. For key establishment, public key cryptography allows different parties to communicate securely without having prior access to a shared secret key, by using one or more pairs (public key and private key) of cryptographic keys.
Public key validation	The procedure whereby the recipient of a public key checks that the key conforms to the arithmetic requirements for such a key in order to thwart certain types of attacks.
Receiver	The party that receives secret keying material via a key transport transaction. Contrast with sender.
Recipient	A party that receives (1) a public key (e.g., in a certificate); (2) assurance, such as an assurance of the validity of a candidate public key or assurance of possession of the private key associated with a public key; or (3) key confirmation. Contrast with provider.
Responder	The party that does not initiate a key agreement transaction. Contrast with initiator.
Scheme	A (cryptographic) scheme consists of a specification of unambiguous transformations that are capable of providing a (cryptographic) service when properly implemented and maintained. A scheme is a higher level construct than a primitive and a lower level construct than a protocol.
Security strength (Also "Bits of security")	A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system.
Security properties	The security features (e.g., entity authentication, replay protection, or key confirmation) that a cryptographic scheme may, or may not, provide.
Sender	The party that sends secret keying material to the receiver using a key transport transaction.
Shall	This term is used to indicate a requirement of a Federal Information processing Standard (FIPS) or a requirement that needs to be fulfilled to claim conformance to this Recommendation. Note that **shall** may be coupled with **not** to become **shall not**.

Shared secret keying material	As used in this Recommendation, the secret keying material that is either (1) derived by applying the key derivation function to the shared secret and other shared information during a key agreement process, or (2) is transported during a key transport process.
Shared secret	A secret value that has been computed during a key establishment scheme and is used as input to a key derivation function to produce keying material.
Should	This term is used to indicate an important recommendation. Ignoring the recommendation could result in undesirable results. Note that **should** may be coupled with **not** to become **should not**.
Symmetric key	A single cryptographic key that is used with a secret (symmetric) key algorithm.
Symmetric key algorithm	A cryptographic algorithm that uses one secret key that is shared between authorized parties.
Target security strength	The desired security strength for a cryptographic application.
Trusted party	A trusted party is a party that is trusted by an entity to faithfully perform certain services for that entity. An entity may choose to act as a trusted party for itself.
Trusted third party	A third party that is trusted by its clients to perform certain services, such as a CA. (By contrast, the initiator and responder (or sender and receiver) in a scheme are considered to be the first and second parties in a key establishment transaction.)

3.2 Symbols and Abbreviations

A	Additional Input that is bound to keying material, a byte string.
$[a, b]$	The set of integers x such that $a \leq x \leq b$.
AES	Advanced Encryption Standard (as specified in FIPS 197 [4]).
ASC	The American National Standards Institute (ANSI) Accredited Standards Committee.
ANS	American National Standard.

ASN.1	Abstract Syntax Notation One.
c	Ciphertext, an integer.
C, C_0, C_1	Ciphertext, each is a byte string.
CA	Certification Authority.
context	Context string for initiator authentication, a bit string.
CRT	Chinese Remainder Theorem.
d	RSA private exponent, an integer.
Data	A variable-length string of zero or more (eight-bit) bytes
dataLen	The length of *Data* in bytes.
DerivedKeyingMaterial	Derived Keying Material, a bit string.
dP	RSA private exponent for the prime factor p in the CRT format, i.e., d mod (p-1), an integer.
dQ	RSA private exponent for the prime factor q in the CRT format, i.e., d mod (q-1), an integer.
e	RSA public exponent, an integer.
eBits	Length in bits of the RSA exponent *e*.
$EphemData_P$, $EphemData_R$, $EphemData_U$, $EphemData_V$	Fresh data contributed by the provider or recipient in a key confirmation; each is a byte string.
GCD(*a*, *b*)	Greatest Common Divisor of two non-negative integers *a* and *b*
H	An **approved** hash function
hBits	Length in bits of a hash value
hLen	Length in bytes of a hash value
HMAC	Keyed-hash Message Authentication Code (as specified in FIPS 198-1 [5])

I2BS	Integer to Byte String
ID	The bit string denoting the identifier associated with an entity.
ID_P, ID_R, ID_U, ID_V	Identifier bit strings for parties P, R, U, and V
IFC	Integer Factorization Cryptography
K	Keying material, a byte string.
KBits	Length in bits of the keying material
KLen	Length in bytes of the keying material
KAS	Key Agreement Scheme.
k	Keying material, an integer.
KAS1-basic	The basic form of Key Agreement Scheme 1
KAS1-responder-confirmation	Key Agreement Scheme 1 with responder-confirmation
KAS2-basic	The basic form of Key Agreement Scheme 2
KAS2-responder-confirmation	Key Agreement Scheme 2 with responder confirmation
KAS2-initiator-confirmation	Key Agreement Scheme 2 with initiator confirmation
KAS2-bilateral-confirmation	Key Agreement Scheme 2 with bilateral confirmation
KC	Key Confirmation
KDF	Key Derivation Function
KEM	Key Encapsulation Mechanism
KeyData	Keying material other than that which is used for the *MacKey* employed in key confirmation.
KTS	Key Transport Scheme (i.e. KTS-OAEP or KTS-KEM-KWS)

KTS-OAEP-basic	The basic form of the Key Transport Scheme with Optimal Asymmetric Encryption Padding
KTS-OAEP-receiver-confirmation	Key Transport Scheme with Optimal Asymmetric Encryption Padding and receiver confirmation
KTS-KEM-KWS-basic	The basic form of the Key Transport Scheme with Key Encapsulation Mechanism and Key-Wrapping Scheme
KTS-KEM-KWS-receiver-confirmation	Key Transport Scheme with Key Encapsulation Mechanism, Key-Wrapping Scheme, and receiver confirmation
KWK	Key-Wrapping Key, a byte string
kwkBits	Length in bits of the key-wrapping key
kwkLen	Length in bytes of the key-wrapping key
KWS	(Symmetric) Key-Wrapping Scheme.
LCM(*a*, *b*)	Least common multiple of two non-negative integers *a* and *b*.
MAC	Message Authentication Code.
MacData	A byte string input to the *MacTag* computation.
$MacData_U$, *(or* $MacData_V$*)*	*MacData* associated with Party *U* (or Party *V,* respectively), and used to generate $MacTag_U$ *(or* $MacTag_V$*,* respectively*)*. Each is a byte string.
MacKey	Key used to compute the MAC, a byte string.
MacKeyLen	Length in bytes of the *MacKey*.
MacTag	A byte string that allows an entity to verify the integrity of the information. *MacTag* is the output of the MAC algorithm. The literature sometimes refers to *MacTag* as a Message Authentication Code (MAC).
$MacTag_V$*,* $(MacTag_U)$	The *MacTag* generated by Party V (or Party U, respectively). Each is a byte string.
MacTagLen	The length of *MacTag* in bytes.

Mask	Mask, a byte string.
maskLen	Length in bytes of the mask.
max_hash_inputBits	An integer that indicates the maximum length, in bits, of a bit string input to the hash function
MGF	Mask Generation Function
mgfSeed	String from which a mask is derived, a byte string.
n	RSA modulus
(*n, d*)	RSA private key in the basic format.
(*n, e*)	RSA public key.
N_V	Nonce contributed by party V, a byte string.
nBits	Length in bits of the RSA modulus *n*.
nLen	Length in bytes of the RSA modulus *n*.
Null	The empty bit string
OtherInfo	Other information for key derivation, a bit string.
p	First prime factor of RSA modulus *n*.
$PrivKey_U$, $PrivKey_V$	Private key of party U or V.
$PubKey_U$, $PubKey_V$	Public key of party U or V.
q	Second prime factor of the RSA modulus *n*.
qInv	Inverse of *q* modulo *p* in the CRT format, i.e., q^{-1} mod *p*, an integer.
RBG	Random Bit Generator.
RSASVE	RSA Secret Value Encapsulation.
RSA-KEM-KWS	RSA Key Encapsulation Mechanism with a Key-Wrapping Scheme

RSA-OAEP	RSA with Optimal Asymmetric Encryption Padding.
S	String of bytes.
s	Security strength in bits.
SHA	Secure Hash Algorithm.
TransportedKeyingMaterial	Transported Keying Material
TTP	A Trusted Third Party.
U	The initiator or sender of a key establishment process.
V	The responder or receiver in a key establishment process.
X	Byte string to be converted to or from an integer, output of conversion from an ASCII string.
$X =? Y$	Verify that *X* equals *Y*.
x	Non-negative integer to be converted to or from a byte string.
$x \bmod n$	The modular reduction of the (arbitrary) integer *x* by the positive integer *n* (the *modulus*). For the purposes of this Recommendation, $y = x \bmod n$ is the unique integer satisfying the following two conditions: $0 \le y < n$ and $x - y$ is divisible by *n*.
$x^{-1} \bmod n$	The multiplicative inverse of the integer *x* modulo the positive integer *n*. This quantity is defined if and only if *x* is relatively prime to *n*. For the purposes of this Recommendation, $y = x^{-1} \bmod n$ is the unique integer satisfying the following two conditions: $0 \le y < n$ and $1 = (xy) \bmod n$.
{*X*}	Indicates that the inclusion of *X* is optional.
{*x*, *y*}	A set containing the integers *x* and *y*.
$X \| Y$	Concatenation of two strings *X* and *Y*.
$\lceil x \rceil$	The ceiling of *x*; the smallest integer ≥ *x*. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.

$\lfloor x \rfloor$	The floor of x, the largest integer less than or equal to x. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
$\lvert x \rvert$	The absolute value of x.
XOR	Exclusive-Or, defined as bit-wise modulo 2 arithmetic with no carry.
Z	A shared secret that is used to derive secret keying material using a key derivation function, a byte string.
z	The integer form of Z
$\lambda(n)$	Lambda function of the RSA modulus n, i.e., the least positive integer i such that $a^i \equiv 1$ for all a relatively prime to n. If $n = pq$ where p and q are distinct primes, then $\lambda(n) = \text{LCM}(p\text{-}1, q\text{-}1)$.
$\phi(n)$	Totient function of the RSA modulus n, i.e., the number of integers between 0 and n-1 that are relatively prime to n. If $n = pq$, where p and q are distinct primes, then $\phi(n) = (p\text{-}1)(q\text{-}1)$.
$\|\|$	Concatenation operator.
$\oplus$	XOR Operator.

4 Key Establishment Schemes Overview

Secret cryptographic keying material may be electronically established between parties by using a key establishment scheme, that is, by using either a key agreement scheme or a key transport scheme.

During key agreement, information is exchanged between both parties that permits each party to contribute to and derive the secret keying material. Key agreement schemes may use either symmetric key or asymmetric key (public key) techniques. The key agreement schemes described in this Recommendation use public key techniques. The party that begins a key agreement scheme is called the initiator, and the other party is called the responder.

During key transport (where one party selects the secret keying material), encrypted secret keying material is transported from the sender to the receiver. The key transport schemes described in this Recommendation use either public key techniques or a combination of public key and symmetric key techniques. The party that sends the secret keying material is called the sender, and the other party is called the receiver.

The security of the Integer Factorization Cryptography (IFC) schemes in this Recommendation is based on the intractability of factoring large integers.

For compliance with this Recommendation, equivalent processes may be used. Two processes are equivalent if, whenever the same values are input to each process (either as input parameters or as values made available during the process), each process produces the same output as the other.

Some processes are used to provide assurance (for example, assurance of the arithmetic validity of a public key or assurance of possession of a private key associated with a public key). The party that provides the assurance is called the provider (of the assurance), and the other party is called the recipient (of the assurance).

Note that the terms initiator, responder, sender, receiver, provider and recipient have specific meanings in this Recommendation.

A number of steps are performed to establish secret keying material as described in Sections 4.1, 4.2, and 4.3.

4.1 Key Establishment Preparations by an Owner

The owner of a private/public key pair is the entity that is authorized to use the private key of that key pair. Figure 1 depicts the steps that may be required of that entity when preparing for a key establishment process (i.e., either key agreement or key transport).

The first step in the process is for the entity to obtain a key pair. Either the entity generates the key pair as specified in Section 6.3 or a trusted third party (TTP) generates the key pair as specified in Section 6.3, and provides it to the entity. The entity (i.e., the owner) obtains assurance of key pair validity, and as part of the process, obtains assurance that it actually possesses the (correct) private key. **Approved** methods for obtaining assurance of key pair validity by the owner are addressed in Section 6.4.1.

An *identifier* (see Section 3.1) is used to name the entity that is authorized to use the private key corresponding to a particular public key (i.e., the identifier names the key pair's owner). This name may uniquely distinguish the entity from all others, in which case it could rightfully be considered an identity. However, the name may be something less specific – an organization, nickname, etc. – hence, the term *identifier* is used in this Recommendation, rather than the term *identity*. A key pair's owner is responsible for ensuring that the identifier associated with its public key is appropriate for the applications in which the public key will be used.

This Recommendation requires that there is a trustworthy binding of each entity's identifier to the entity's public key. The binding of an identifier to a public key may be accomplished by a trusted authority (i.e., a binding authority; for example, a registration authority working with a CA who creates a certificate containing both the public key and the identifier). The binding authority verifies the identifier chosen for the owner. The binding authority is also responsible for checking the arithmetic validity of the owner's public key, and the owner's possession of the private key corresponding to that public key. The methods used by a third party trusted by the

recipient to obtain that assurance are beyond the scope of this Recommendation (see Seon 8.1.5.1.1.2 of SP 800-57 [7]).

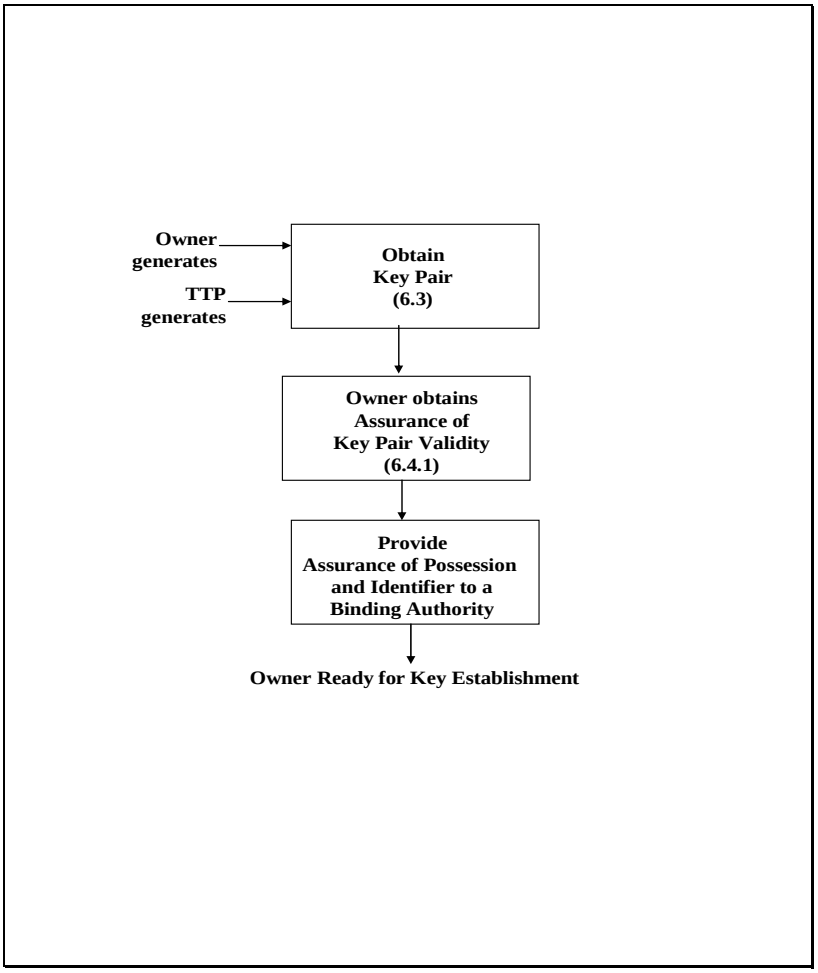

Figure 1: Owner Key Establishment Preparations

After the above steps have been performed, the entity (i.e., the key pair owner) is ready to enter into a key establishment process.

4.2 Key Agreement Process

Figure 2 depicts the steps implemented by an entity when establishing secret keying material with another entity using one of the key agreement schemes described in this Recommendation. (Some discrepancies in ordering may occur in practice, depending on the communication protocol in which the key agreement process is performed.) Depending on the key agreement scheme, the entity could be either the key agreement initiator or responder. Note that some of the actions shown may not be a part of every scheme. For example, key confirmation is not provided in the basic key agreement schemes (see Sections 8.2.2 and 8.3.2). The specifications of this recommendation indicate when a particular action is required.

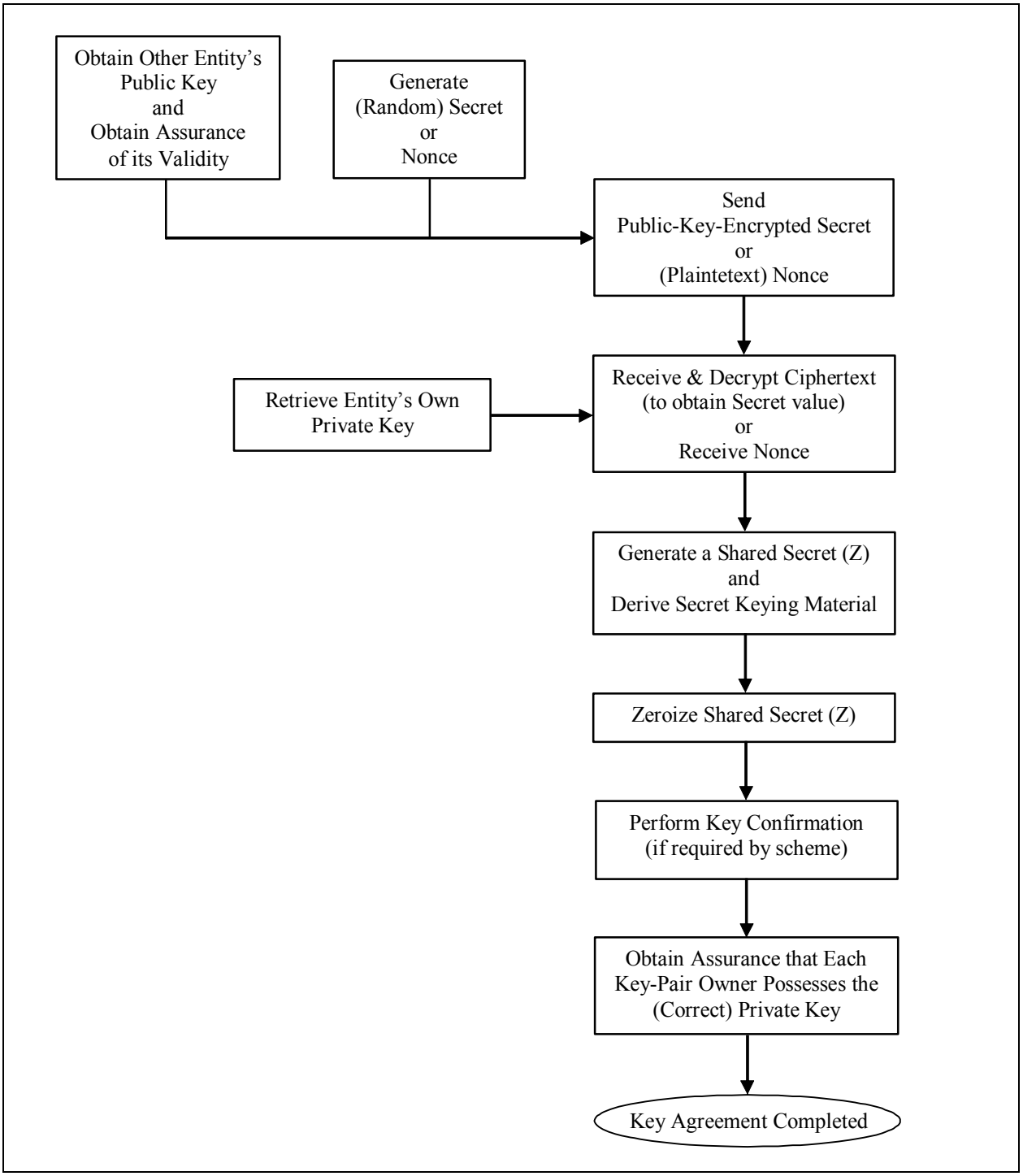

Figure 2: Key Agreement Process

Each participant obtains the identifier associated with the other entity, and verifies that the identifier of the other entity corresponds to the entity with whom the participant wishes to establish secret keying material.

Each entity that requires the other entity's public key for use in the key agreement scheme obtains the public key bound to the other party's identifier, and obtains assurance of the validity of the public key. **Approved** methods for obtaining assurance of the validity of another entity's public key are provided in Section 6.4.2.

Each entity generates either a (random) secret value (which becomes a shared secret when transmitted to the other entity) or a nonce, as required by the particular key agreement scheme. If the scheme requires an entity to generate a secret value, that secret value is generated as specified in Section 5.3 and encrypted using the other entity's public key. The resulting ciphertext is then provided to the other entity. If the key agreement scheme requires that an entity provide a nonce, that nonce is generated as specified in Section 5.6 and provided (in plaintext form) to the other party. (See Sections 8.2 and 8.3 for details).

Each participant in the key agreement process uses the appropriate public and/or private keys to establish a shared secret (Z) as specified Section 8.2.2 or 8.3.2. Each participant then derives secret keying material from the shared secret (and other information), as specified in Section 5.9.

If the key agreement scheme includes key confirmation provided by one or both of the participants, then key confirmation is performed as specified in Section 8.2.3 or 8.3.3, thus providing assurance that the key pair owner possesses the (correct) private key.

The owner of any key pair used during the key agreement transaction is required to have assurance that the owner is in possession of the correct private key. Likewise, the recipient of another entity's public key is required to have assurance that its owner is in possession of the corresponding private key. Assurance of private key possession is obtained prior to using the derived keying material for purposes beyond those of the key agreement transaction itself. This assurance may be provided/obtained either through key confirmation, or by some other **approved** means (see Sections 6.5.1 and 6.5.2).

4.3 IFC-based Key Transport Process

Figure 3 depicts the steps implemented by two entities when using one of the key-transport schemes described in this Recommendation to establish secret keying material.

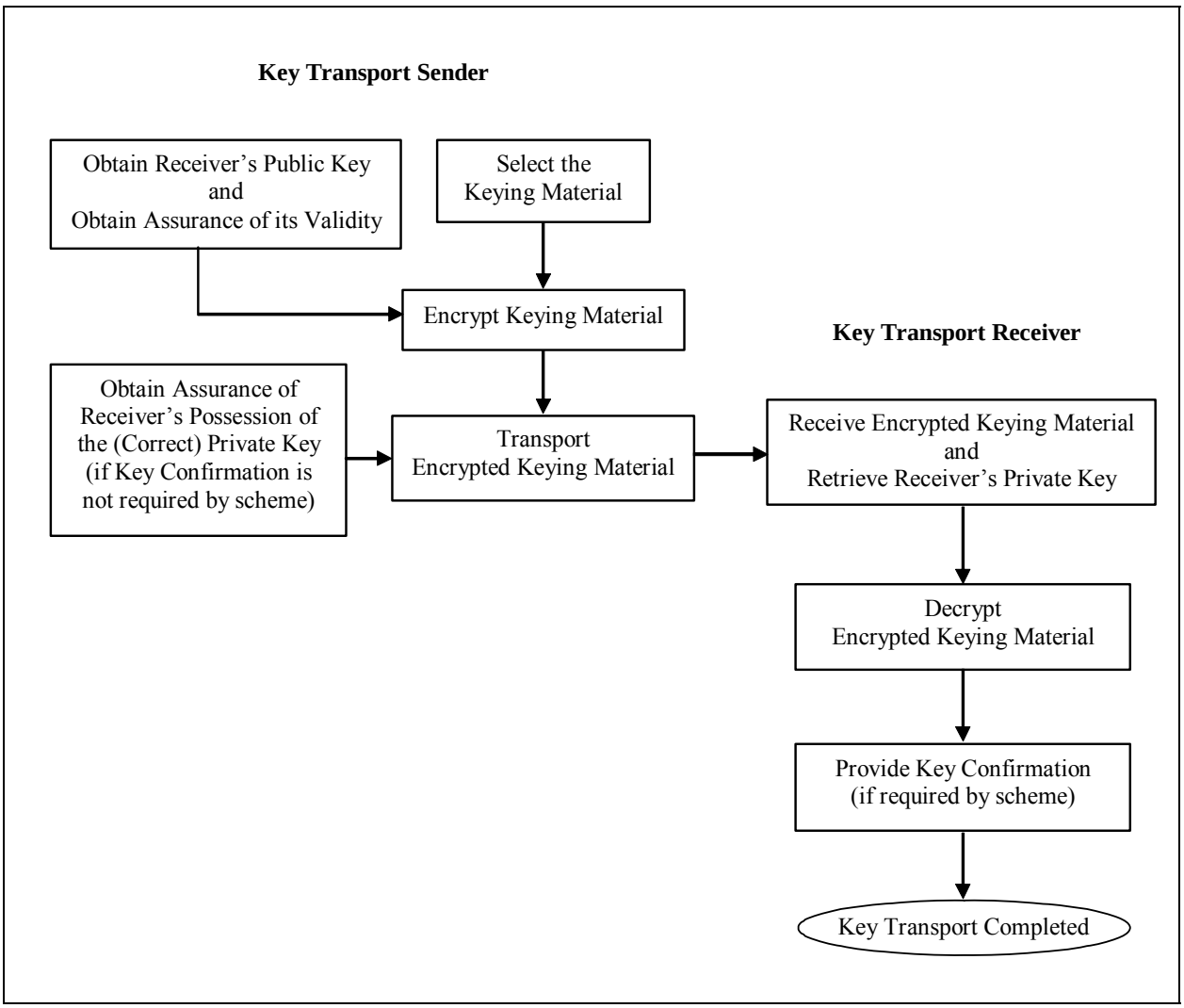

Figure 3: Key Transport Process

The entity who will act as the sender obtains the identifier associated with the entity that will act as the receiver, and verifies that the receiver's identifier corresponds to an entity with whom the sender wishes to establish secret keying material.

Prior to performing key transport, the sender obtains the receiver's public key and obtains assurance of its validity. **Approved** methods for obtaining assurance of the validity of another entity's public key are provided in Section 6.4.2. The sender is also required to have assurance that the receiver is in possession of the private key corresponding to the receiver's public key prior to key transport, unless that assurance is obtained via key confirmation included as part of the scheme. (See Sections 9.2 and 9.3 for details).

The sender selects the secret keying material (and, perhaps, other data) to be transported to the other entity. Then, using the intended receiver's public key, the sender either encrypts that material directly (as specified in Section 9.2.3), or, employs a combination of secret value encapsulation and key-wrapping (as specified in Section 9.3.3). The resulting ciphertext is transported to the receiver.

Prior to participating in a key establishment transaction, the receiver is required to have assurance of the validity of it's key pair. This assurance may be renewed whenever fresh assurance is desired. Upon (or before) receipt of the transported ciphertext, the receiver retrieves the private key from its own key pair. Using its private key, the receiver takes the necessary steps (as specified in Section 9.2.3 or 9.3.3) to decrypt the ciphertext and obtain the plaintext keying material.

If the key-transport scheme includes key confirmation, then key confirmation is provided by the receiver to the sender as specified in Section 9.2.4 or 9.3.4. Through the use of key confirmation, the sender can obtain assurance that the receiver has correctly recovered the keying material from the ciphertext. The sender can also obtain assurance that the receiver was in possession of the correct private key.

5 Cryptographic Elements

This section describes the cryptographic elements that are used in the development of key establishment schemes.

5.1 Cryptographic Hash Functions

An **approved** hash function **shall** be used when a hash function is required (for example, for the key derivation function or to compute a MAC when HMAC, as specified in FIPS 198-1 [5], is used). FIPS 180-3 [2] specifies **approved** hash functions. The hash function **shall** be selected in accordance with the parameter lists in Table 1 of Section 6.2.3.

5.2 Message Authentication Code (MAC) Algorithm

A Message Authentication Code (MAC) algorithm defines a family of one-way (MAC) functions that is parameterized by a symmetric key. The MAC algorithm is used to provide key confirmation as specified in this Recommendation using an appropriate scheme from this Recommendation, and is used to validate implementations of the key establishment schemes specified in this Recommendation (see Section 5.2.3).

In the case of key confirmation, an entity is required to compute a *MacTag* on received or derived data using the MAC function determined by a symmetric key derived from a shared secret (when a key agreement scheme is used) or from transported keying material (when a key transport scheme is used). The *MacTag* is sent to another entity in order to confirm that the keying material is correct. An **approved** MAC algorithm with appropriate parameter choices (see Section 6.2.3) **shall** be used to compute a *MacTag*, for example, HMAC [5] or CMAC[6].

5.2.1 *MacTag* Computation

The computation of the *MacTag* is represented as follows:

$$MacTag = \text{MAC}(MacKey, MacTagLen, MacData).$$

The *MacTag* computation **shall** be performed using an **approved** MAC algorithm. In the above equation, MAC represents an **approved** MAC algorithm; *MacKey* represents a symmetric key obtained from the *DerivedKeyingMaterial* (when a key agreement scheme is used for key confirmation) or from the transported keying material (when a key transport scheme is used for key confirmation) (see Section 6.6.1 along with Sections 8.2.3 and 8.3.3 for key agreement and 9.2.4 and 9.3.4 for key transport); *MacTagLen* represents the length of *MacTag*; and *MacData* represents the data on which the *MacTag* is computed. The minimum for *MacTagLen* is specified in Table 1 of Section 6.2.3. The minimum length for *MacKey* is also specified in Table 1. See [5] and [6].

5.2.2 *MacTag* Checking

To check a received *MacTag* (e.g., received during key confirmation and/or implementation validation), a new *MacTag* is computed—using the values of *MacKey*, *MacTagLen*, and *MacData* possessed by the recipient/receiver (as specified in Sections 5.2.1 and 5.2.3). The new *MacTag* is compared with the received *MacTag*. If their values are equal, then it may be inferred that the same *MacKey*, *MacTagLen*, and *MacData* values were used in the two *MacTag* computations.

5.2.3 Implementation Validation Message

For purposes of validating an implementation of the schemes in this Recommendation during an implementation validation test (under the NIST Cryptographic Algorithm Validation Program), the value of *MacData* **shall** be the string "Standard Test Message", followed by a 128-bit field for a nonce. The default value for this field is all binary zeros. Different values for this field will be specified during testing. This is for the purpose of testing when no key confirmation capability exists.

5.3 Random Bit Generation

Whenever this Recommendation requires the use of a randomly generated value (for example, for keys or nonces), the values **shall** be generated using an **approved** random bit generator (RBG) at an appropriate security strength. **Approved** RBG methods and methods for converting the random bits to an integer are provided in SP 800-90. The security strength provided by an RBG employed in this Standard **shall** be greater than or equal to the target security strength for the scheme in which it is employed.

5.4 Prime Number Generators

A prime number generator employs a random bit generator and a primality test in order to produce random prime numbers in a certain range, possibly with certain structure.

Only **approved** prime number generation methods **shall** be employed in this Recommendation. **Approved** prime number generation methods are specified in FIPS 186-3 [3].

The prime number generators in FIPS 186-3 accept input values that include the selected public exponent *e* and the desired length (in bits) of the modulus *n*, and use an **approved** random bit generator to generate the prime factors *p* and *q* with (at least) the following properties:

1. Each is between $\lceil \sqrt{2}(2^{nBits/2-1}) \rceil$ and $2^{nBits/2} - 1$, inclusive,
2. $p - 1$ and $q - 1$ are relatively prime to *e*, and
3. $|p - q| \leq 2^{nBits/2-100}$.

5.5 Primality Testing Methods

A primality testing method determines whether an integer is prime with a negligible probability of error. Only **approved** primality testing methods **shall** be employed in this Standard. **approved** primality testing methods as of the publication of this Standard are listed in FIPS 186-3.

5.6 Nonces

A nonce is a time-varying value, represented as a byte string that has (at most) a negligible chance of repeating. For example, a nonce may be composed of one (or more) of the following components:

1. A random value that is generated anew for each nonce, using an **approved** random bit generator. The security strength of the RBG used to obtain each random value **shall** be greater than or equal to the security strength associated with the modulus used in the key establishment scheme (see SP 800-57-Part 1 [7]). The length of the RBG output **shall** be at least the security strength of the key establishment scheme. A nonce containing a component of this type is called a *random nonce*.
2. A timestamp of sufficient resolution (detail) so that it is different each time it is used.
3. A monotonically increasing sequence number.

If a combination of a timestamp and a monotonically increasing sequence number is used without a random nonce, the sequence number **shall** be reset only when the timestamp changes. (For example, a timestamp may show the date but not the time of day, so a sequence number is appended that will not repeat during a particular day.)

Nonces are used, for example, in implementation validation testing (see Section 5.2.3), and in KAS1 schemes (see Section 8.2).

When using a nonce, a random nonce **should** be used.

5.7 Symmetric Key-Wrapping Algorithms

A symmetric key-wrapping algorithm wraps (i.e., encrypts and integrity-protects) keying material using a symmetric key-wrapping key. In this Recommendation, a symmetric key-wrapping algorithm is used by the KTS-KEM-KWS schemes specified in Section 9.3. The wrapping operation produces a ciphertext C from keying material K, using the key-wrapping key KWK and additional input A which is known to both the wrapping and the unwrapping parties, but may be null. A is bound to K in that C is a cryptographic function of both values. The unwrapping operation recovers K from C using KWK and A; the unwrapping operation then verifies the integrity of K and A by means of an integrity test built into the wrapping algorithm. Thus, the key-wrapping algorithm **shall** support both confidentiality and integrity properties. There may be restrictions on the length of the keying material and the additional input, but such bounds are generally very large.

In this Recommendation, the wrapping operation is specified as:

C = KWA.WRAP(KWK, K, A) ,

and the unwrapping operation is specified as:

K = KWA.UNWRAP(KWK, C, A) ,

where *KWK* is the key-wrapping key, *K* is the plaintext keying material, *A* is additional input, and *C* is the ciphertext.

Approved/allowed[1] key-wrapping algorithms **shall** be used that employ **approved** block cipher algorithms and keys that support a security strength that is equal to or greater than the security strength required to protect the data to be cryptographically protected by the wrapped keying material (see [7]). For example, if the data requires 112 bits of security (the target security strength), the block cipher and keys used for key-wrapping **shall** support a security strength of at least 112 bits. Note, that in this case, the wrapped keying material, together with the algorithm to be used to protect the data, **shall** also support a security strength of at least 112 bits.

5.8 Mask Generation Function (MGF)

MGF is a mask generation function based on an **approved** hash function (see Section 5.1). The purpose of the MGF is to generate a string of bits that may be used to "mask" other bit strings. The MGF is used by the RSA-OAEP based schemes specified in Section 9.2. The lengths of the MGF seed and the mask in MGF are both variable.

Let H be an **approved** hash function, and let *hLen* denote the length of the hash function output in bytes.

For the purposes of this Standard, MGF **shall not** be run more than once by each party during a given transaction, using a given MGF seed (i.e., a mask **shall** be derived at most once from a given MGF seed).

Function call: MGF(*mgfSeed*, *maskLen*)

Input:

1. *mgfSeed*: a string from which the mask is generated, a byte string.
2. *maskLen*: the intended length in bytes of the mask.

Output:

mask: a byte string of length *maskLen* bytes.

Errors:

An indication that the mask is too long.

---

[1] Allowed key wrap algorithms are specified in FIPS 140-2 Implementation Guidance IG 7.1.

Process:

1. Let *T* be the empty string.
2. For *counter* from 0 to $\lceil maskLen / hLen \rceil - 1$, do the following:
   a. Let *D* = I2BS(*counter*, 4) (see Appendix B.1).
   b. Let *T* = *T* || H(*mgfSeed* || *D*).
3. Output the first *maskLen* bytes of *T* as the byte string *mask*.

5.9 Key Derivation Functions for Key Establishment Schemes

An **approved** or allowed (i.e., see FIPS 140-2/3 Annexes) key derivation function (KDF) **shall** be used to derive secret keying material from a shared secret during the execution of any key establishment scheme from the KAS1, KAS2, and KTS-KEM-KWS families of schemes. The output from the KDF **shall** only be used for secret keying material, such as a symmetric key used for data encryption or message integrity, a secret initialization vector, or a master key that will be used to derive other keys (possibly using a different process). Non-secret keying material (such as a non-secret initialization vector) **shall not** be generated using the shared secret.

Each call to the KDF requires a freshly computed shared secret, and this shared secret **shall** be zeroized immediately following its use. The derived secret keying material **shall** be computed in its entirety before outputting any portion of it.

The derived secret keying material may be parsed into one or more keys or other secret cryptographic keying material (for example, secret initialization vectors). In cases where key confirmation is included in a key agreement scheme from the KAS1 family or the KAS2 family, *MacKey* **shall** be formed from the initial bits of the KDF output. (When key confirmation is included in a key transport scheme from the KTS-OAEP family or the KTS-KEM-KWS family, *MacKey* is not obtained from the output of the KDF, but **shall** be formed from the initial bits of the transported keying material.) In all cases, *MacKey* **shall** be zeroized after its use (in particular, *MacKey* **shall not** be used for purposes other than key confirmation).

Sections 5.9.1 and 5.9.2 specify two **approved** KDFs for use in key establishment. They differ only in the way that they format the *OtherInfo* bit string. Other allowable methods and the protocols that they may be used with are referenced in FIPS 140-2 Annex D. Any hash function used in a KDF **shall** be **approved** (see Section 5.1) and **shall** also meet the selection requirements specified herein (see Table 1 in Section 6.2.3).

5.9.1 Concatenation Key Derivation Function (Approved Alternative 1)

This section specifies an **approved** key derivation function, based on concatenation.

The Concatenation KDF is as follows:

Function call: KDF(*Z, KBits, OtherInfo*).

Fixed Values (implementation dependent):

1. *hBits*: an integer that indicates the length (in bits) of the output of the hash function used to derive the secret keying material.
2. *max_hash_inputBits*: an integer that indicates the maximum length (in bits) of a bit string input to the hash function.

Auxiliary Function:

H: an **approved** hash function.

Input:

1. *Z*: a byte string that is the shared secret.
2. *KBits*: An integer that indicates the length (in bits) of the secret keying material to be generated; *KBits* **shall** be less than or equal to $hBits \times (2^{32} - 1)$.
3. *OtherInfo*: A bit string equal to the following concatenation:

   AlgorithmID || *PartyUInfo* || *PartyVInfo*{ || *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 after any *MacKey* is extracted from the derived keying material when key confirmation is performed. For example if key confirmation is not performed, *AlgorithmID* might indicate that bits 1-128 are to be used as a 128-bit AES key. If key confirmation is performed, then *AlgorithmID* might indicate that bits 1- 128 are used as a *MacKey,* and bits 129-256 are to be used as the 128-bit AES key.

   b. *PartyUInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by party U to the key derivation process. At a minimum, *PartyUInfo* **shall** include ID_U, the identifier of party U, as a separate unit of information.

   c. *PartyVInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by party V to the key derivation process. At a minimum, *PartyVInfo* **shall** include ID_V, the identifier of party V, as a separate unit of information. When this KDF is used in a KAS1 scheme, the nonce, N_V, supplied by party V **shall** be included in *PartyVInfo* as a separate unit of information, immediately following ID_V.

   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*, *PartyUInfo*, and *PartyVInfo* **shall** be the concatenation of a fixed, application specific 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 (eight-bit) 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*, *PartyUInfo*, and *PartyVInfo*, 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 a fixed, application specific sequence of substrings of additional information that may be used in key derivation upon mutual agreement of parties U and V. 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 parties U and V 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 substrings that may be used in the allowed subfield(s) and **shall** specify the fixed-length of the *dataLen* counters.

Output:

The bit string *DerivedKeyingMaterial* of length *KBits* bits (or an error indicator). Any scheme attempting to call this key derivation function with *KBits* greater than or equal to $hBits \times (2^{32} - 1)$ **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_inputBits* bits long **shall** cause the KDF to output an error indicator and stop without outputting *DerivedKeyingMaterial*.

Process:

1. $reps = \lceil KBits / hBtis \rceil$.
2. If $reps > (2^{32} - 1)$, then output an error indicator and stop.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16}.
4. If *counter* || *Z* || *OtherInfo* is more than *max_hash_inputBits* bits long, then output an error indicator and stop.
5. For $i = 1$ to *reps* by 1, do the following:
    a. Compute $Hash_i = H(counter \| Z \| OtherInfo)$.

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

6. Let *Hhash* be set to $Hash_{reps}$ if (*KBits* / *hBits*) is an integer; otherwise, let *Hhash* be set to the (*KBits* mod *hBits*) leftmost bits of $Hash_{reps}$.
7. Set $DerivedKeyingMaterial = Hash_1 \| Hash_2 \| \ldots \| Hash_{reps-1} \| Hhash$.

Notes:

1. Party U **shall** be the initiator or sender, and party V **shall** be the responder or receiver, as assigned by the relying protocol in accordance with the use of those designators by the key establishment scheme employing the KDF.
2. When a party owns a key pair that is used by the key establishment scheme, the identifier assigned to that party **shall** be one that is bound to that key pair. (This will always be the case for party V.) If a key establishment scheme does not require a party to contribute a public key, then the identifier of that party is a non-null identifier selected in accordance with the protocol utilizing the scheme (This may be the case for party U.). The rationale for including the identifiers in the KDF input is provided in Appendix B of [SP 800-56A].

5.9.2 ASN.1 Key Derivation Function (Approved Alternative 2)

This section specifies an **approved** key derivation function utilizing ASN.1 DER encoding of *OtherInfo*. In all other respects, it is the same as the key derivation function specified in Section 5.9.1.

The ASN.1 KDF is as follows:

Function call: KDF(*Z, KBits, OtherInfo*).

Fixed Values (implementation dependent):

1. *hBits*: an integer that indicates the length (in bits) of the output of the hash function used to derive the secret keying material.
2. *max_hash_inputBits*: an integer that indicates the maximum length (in bits) of a bit string input to the hash function.

Auxiliary Function:

H: an **approved** hash function.

Input:

1. *Z*: a byte string that is the shared secret.
2. *KBits*: An integer that indicates the length (in bits) of the secret keying material to be generated; *KBits* **shall** be less than or equal to $hBits \times (2^{32} - 1)$.
3. *OtherInfo*: A bit string specified in ASN.1 DER encoding, which consists of the following subfields of information in some application-specific order:

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 after any *MacKey* is extracted from the derived keying material when key confirmation is performed. For example if key confirmation is not performed, *AlgorithmID* might indicate that bits 1-128 are to be used as a 128-bit AES key. If key confirmation is performed, then *AlgorithmID* might indicate that bits 1- 128 are used as a *MacKey,* and bits 129-256 are to be used as the 128-bit AES key.

b. *PartyUInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by party U to the key derivation process. At a minimum, *PartyUInfo* **shall** include ID_U, the identifier of party U, as a separate unit of information.

c. *PartyVInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by party V to the key derivation process. At a minimum, *PartyVInfo* **shall** include ID_V, the identifier of party V, as a separate unit of information. When this KDF is used in a KAS1 scheme, the nonce, N_V, supplied by party V **shall** be included in *PartyVInfo* as a separate unit of information, following ID_V.

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).

An application using this KDF is responsible for specifying the ASN.1 structure of *OtherInfo*. In particular, applications using this KDF **shall** specify the ordering, number, and ASN.1 type of the separate units of information contained in each of the subfields *AlgorithmID*, *PartyUInfo*, and *PartyVInfo*. Applications allowing the use *SuppPrivInfo* subfield and/or the *SuppPubInfo subfield*, **shall** also specify the ordering, number and ASN.1 type of the additional units of information that may be used in the allowed subfield(s).

Output:

The *DerivedKeyingMaterial* as a bit string of length *KBits* bits (or an appropriate error indicator). The ASN.1 KDF produces secret keying material that is at most $hBits \times (2^{32} - 1)$ bits in length. Any call to this key derivation function using a *KBits* value that is greater than $hBits \times (2^{32} - 1)$ 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_inputBits* bits long shall cause the KDF to output an error indicator and stop without outputting *DerivedKeyingMaterial*.

Process:

1. $reps = \lceil KBits / hBits \rceil$.
2. If $reps > (2^{32} - 1)$, then output an error indicator and stop.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16}.
4. If *counter* || *Z* || *OtherInfo* is more than *max_hash_inputBits* bits long, then output an error indicator and stop.
5. For $i = 1$ to *reps* by 1, do the following:
    a. Compute $Hash_i = H(counter \| Z \| OtherInfo)$.
    b. Increment *counter* (modulo 2^{32}), treating it as an unsigned 32-bit integer.
6. Let *Hhash* be set to $Hash_{reps}$ if (*KBits* / *hBits*) is an integer; otherwise, let *Hhash* be set to the (*KBits* mod *hBits*) leftmost bits of $Hash_{reps}$.
7. Set DerivedKeyingMaterial = $Hash_1 \| Hash_2 \| \ldots \| Hash_{reps-1} \| Hhash$.

Notes:

1. Party U **shall** be the initiator or sender, and party V **shall** be the responder or receiver, as assigned by the relying protocol in accordance with the use of those designators by the key establishment scheme employing the KDF.
2. When a party owns a key pair that is used by the key establishment scheme, the identifier assigned to that party **shall** be one that is bound to that key pair. (This will always be the case for party V.) If a key establishment scheme does not require a party to contribute a public key, then the identifier of that party is a non-null identifier selected in accordance with the protocol utilizing the scheme. (This may be the case for party U.). The rationale for including the identifiers in the KDF input is provided in Appendix B of [SP 800-56A].

6 RSA Key Pairs

6.1 General Requirements

The following are requirements on key pairs (see the Recommendation for Key Management [7]):

1. Each key pair **shall** be created using an **approved** key generation method as specified in Section 6.3.
2. The private keys and prime factors **shall** be protected from unauthorized access, disclosure, and modification.
3. Each Public key-establishment key **shall be** bound to an identifier corresponding to the owner.

4. Public keys **shall** be protected from unauthorized modification. This is often accomplished by using public key certificates that have been signed by a Certification Authority (CA).

5. A recipient of a public key **shall** be assured of the data integrity and correct association of (a) the public key and (b) the identifier of the entity that owns the key pair (that is, the party with whom the recipient intends to establish secret keying material). This assurance is often provided by verifying a public-key certificate that was signed by a trusted third party (for example, a CA), but may be provided by direct distribution of the public key and identifier from the owner, provided that the recipient trusts the owner and distribution process to do this.

6. One key pair **shall not** be used for different cryptographic purposes (for example, a digital signature key pair **shall not** to be used for key establishment or vice versa) with the following possible exception: when requesting the (initial) certificate for a public key-establishment key, the key establishment private key associated with the public key may be used to sign the certificate request. A key pair may be used in more than one key establishment scheme. However, a key pair use for schemes specified in this recommendation **should not** be used for any schemes not specified herein.

7. An owner and a recipient of a public key **shall** have assurance of the validity of the owner's public key. This assurance may be provided, for example, through the use of a public key certificate if the CA obtains sufficient assurance of public key validity as part of its certification process. See Section 6.4. The application performing the key establishment on behalf of the recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself.

8. An owner and a recipient of a public key **shall** have assurance of the owner's possession of the associated private key (see Section 6.5). The owner or a process acting on behalf of the owner **shall** know the method used to obtain assurance of possession of the owner's private key. The recipient or process acting on behalf of the recipient **shall** know the method used to provide assurance to the recipient of the owner's possession of the private key. This assurance may be provided, for example, through the use of a public key certificate if the CA obtains sufficient assurance of possession as part of its certification process.

9. The owner shall have assurance of the validity and possession of the owner's key pair (See Sections 6.4.1 and 6.5.1). (Make a higher number)

6.2 Criteria for RSA Key Pairs for Key Establishment

6.2.1 Definition of a Key Pair

An RSA key pair, in its basic form, consists of an RSA public key (n, e) and an RSA private key (n, d), where:

1. *n*, the modulus, **shall** be the product of exactly two odd positive prime factors, *p* and *q* where *nBits* is the length in bits of *n* as specified for the desired security strength *s* (see Table 1) and *nLen* is the corresponding length in bytes.

2. The public exponent *e* **shall** be selected with the following constraints:

   a. The public exponent *e* **shall** be selected prior to generating the prime factors *p* and *q* and the private exponent *d*.

   b. The exponent *e* **shall** be an odd positive integer such that:

      $65{,}537 \leq e < 2^{256}$

      Note that the value of *e* may be the same for different key pairs.

3. Two secret and randomly generated positive primes *p* and *q* **shall** be selected with the following constraints:

   a. The prime factors of the modulus **shall** be generated independently at random for different key pairs.

   b. LCM((*p*-1), (*q*-1)) **shall** be greater than *e* and relatively prime to e.

   c. The private prime factor *p* **shall** be selected randomly from the primes that satisfy $(\sqrt{2})(2^{(nBits/2)-1}) \leq p \leq (2^{nBits/2}-1)$.

   d. The private prime factor *q* **shall** be selected randomly from the primes that satisfy $(\sqrt{2})(2^{(nBits/2)-1}) \leq q \leq (2^{nBits/2}-1)$.

   e. The difference between *p* and *q* **shall** be $> 2^{(nBits/2)-100}$.

   f. The prime factors *p* and *q* **shall** be generated using an **approved** method meeting the above constraints. Such methods are provided in Appendix B.3 of FIPS 186-3.

4. The private exponent *d* **shall** be selected with the following constraints after the generation of *p* and *q*:

   a. The exponent *d* **shall** be a positive integer value such that $2^{nBits/2} < d <$ LCM((*p*-1), (*q*-1)), and

   b. $1 = ed \bmod$ LCM((*p*-1), (*q*-1)). (That is, $d = e^{-1} \bmod$ (LCM((*p*-1), (*q*-1))).

In the extremely rare event that $d \leq 2^{nbits/2}$, then new values for *p*, *q*, and *d* **shall** be determined and a different value of *e* may be used.

To generate key pairs meeting the above requirements see [3].

6.2.2 Formats

Note that the RSA private key may be expressed in several formats. The basic format of the RSA private key consists of the modulus *n* and a private key exponent *d* that depends on *n* and the public key exponent *e*; this format is used throughout this Recommendation. The other two formats may be used in implementations but may require appropriate modifications for correct implementation. To facilitate implementation testing, the private key **shall** be one of the following:

1. The basic format: (*n*, *d*).
2. The prime factor format: (*p*, *q*, *d*).
3. The Chinese Remainder Theorem (CRT) format: (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*), where $dP = d \bmod (p - 1)$, $dQ = d \bmod (q - 1)$, and $qInv = q^{-1} \bmod p$.

Key pair generators and key pair validation methods are given for each of these formats in Section 6.3.

6.2.3 Parameter Length Sets

Federal Government entities **shall** select a target security strength for the scheme of either 80 bits or 112 bits and select scheme parameters from the target security strength as shown in Table 1. The target security strength **shall** be selected based on the security needs of the information that will be protected using the keying material agreed upon using this Recommendation. SP 800-57 [7] deprecates the use of the 80 bit security strength after 2010. Entities **shall** select a target security strength of 112 bits if the security life of the information extends beyond 2010. This Recommendation specifies two choices for the modulus bit length: 1024 and 2048 bits. The security strength of the modulus bit length **shall** meet or exceed that of the target security strength of the scheme. See the comparable strengths table in SP 800-57 to assess the comparable security strength of a particular modulus bit length.

Implementations which include a hash function (e.g. for use in a key derivation function or RSA-OAEP) **shall** select any **approved** hash function.

Implementations which include a MAC algorithm (e.g. for key confirmation) **shall** employ an **approved** MAC algorithm. The *MacKey* length **shall** meet or exceed the target security strength, and **should** meet or exceed the security strength of the modulus. The *MacTag* length **shall** meet or exceed 64 bits (8 bytes).

For example, an entity may select the 112 bit target security strength for an application which establishes a 128 bit AES key. An implementation for the scheme may select 2048 bits, SHA-256, and HMAC-SHA-256 for the modulus length, hash function and MAC algorithm, along with *MacTag* and *MacKey* lengths of 64 bits and 128 bits, respectively.

Table 1: IFC Parameters for Key Establishment

IFC Parameter Name	Target Security Strength	
	80 bits	**112 bits**
Bit length of *n*	1024 bits or 2048 bits	2048 bits
Minimum MacKey length	80 bits/10 bytes	112 bits/14 bytes
Minimum *MacTag* length	64 bits/8 bytes	64 bits/8 bytes

6.3 RSA Key Pair Generators

An RSA key pair generator produces a random RSA public-key/private-key pair at a target security strength, given an appropriate RSA key length and possibly other inputs. A key pair generator requires a random bit generator (RBG) and a prime number generator. **Approved** prime number generators may place additional constraints on RSA key pair generation, depending on the target security strength (see FIPS 186-3 [3]). Approved RSA key pair generators **shall** be employed. For key pair criteria, see Section 6.2.1.

6.3.1 RSAKPG1 Family: RSA Key Pair Generation with a Fixed Public Exponent

The RSAKPG1 family consists of three RSA key pair generators where the public exponent has a fixed value (see Section 6.2).

Three representations are addressed:

1. *rsakpg1-basic* generates the private key in the basic format (*n*, *d*),
2. *rsakpg1-prime-factor* generates the private key in the prime factor format (*p*, *q*, *d*), and
3. *rsakpg1-crt* generates the private key in the Chinese Remainder Theorem format (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*).

An implementation may perform a key pair validation before outputting the key pair from the generator. The key pair validation methods for this family are specified in Section 6.4.1.2.

6.3.1.1 rsakpg1-basic

rsakpg1-basic is the generator in the RSAKPG1 family where the private key is in the basic format (*n*, *d*).

Function call: *rsakpg1-basic*(*s*, *nBits*, *e*)

Input:

1. *s*: the target security strength for the key-pair generator, an integer in the set {80, 112},
2. *nBits*: the intended length in bits of the RSA modulus, an integer (see Table 1), and
3. *e*: a fixed public exponent, an odd integer such that $65{,}537 \le e < 2^{256}$.

Output:

1. (*n*, *e*): the RSA public key, and
2. (*n*, *d*): the RSA private key in the basic format.

Error: Indications of the following:

1. The security strength is out of range,
2. The modulus length is out of range,
3. The fixed public exponent is out of range, or
4. Pair-wise consistency failure.

Process:

1. Check the ranges:
    a. If *s* is not an integer in the set {80, 112}, output an indication that the security strength is out of range and stop.
    b. If *nBits* is not an integer in the set {1024, 2048}, or if *nBits* is less than the minimum key length for the target security strength, *s*, output an indication that the modulus length is out of range and stop.
    c. If *e* is not an odd integer such that $65{,}537 \le e < 2^{256}$, output an indication that the exponent is out of range and stop.
2. Generate the prime factors *p* and *q* (see Section 5.4).
3. Determine the private exponent *d*:

$$d = e^{-1} \bmod \mathrm{LCM}(p-1, q-1) .$$

    In the very rare event that $d \le 2^{nBits/2}$, discard *d*, and repeat the process, starting at step 2.

4. Determine the modulus *n* as $n = p \cdot q$.

5. Perform a pair-wise consistency test by verifying that $k = (k^e)^d \bmod n$ for some integer k satisfying $1 < k < n$. If an inconsistency is found, output an indication of a pair-wise consistency failure and stop.
6. Output (n, e) as the public key, and (n, d) as the private key.

Note that key pair validation as specified in Section 6.4.1.2.1 can be performed after step 5 and before step 6. If an error is detected, output an indication of key pair validation failure and stop.

6.3.1.2 rsakpg1-prime-factor

rsakpg1-prime-factor is the generator in the RSAKPG1 family where the private key is in the prime factor format (p, q, d).

Function call: *rsakpg1-prime-factor*(s, *nBits*, e)

The inputs, outputs and errors are the same as in *rsakpg1-basic* (see 6.3.1.1), except that the private key is in the prime factor format: (p, q, d).

The steps are the same as in *rsakpg1-basic***,** except that processing Step 6 is replaced by the following:

6. Output (n, e) as the public key, and (p, q, d) as the private key.

Note that key pair validation as specified in Section 6.4.1.2.2 can be performed after step 5 and before step 6. If an error is detected, output an indication of key pair validation failure and stop.

6.3.1.3 rsakpg1-crt

rsakpg1-crt is the generator in the RSAKPG1 family where the private key is in the Chinese Remainder Theorem format (n, e, d, p, q, dP, dQ, $qInv$).

Function call: *rsakpg1-crt*(s, *nBits*, e)

The inputs, outputs and errors are the same as in *rsakpg1-basic* (see 6.3.1.1), except that the private key is in the Chinese Remainder Theorem format: (n, e, d, p, q, dP, dQ, $qInv$).

The steps are the same as in *rsakpg1-basic***,** except that processing Steps 5 and 6 are replaced by the following:

5. Determine the components dP, dQ and $qInv$:
    a. $dP = d \bmod (p - 1)$.
    b. $dQ = d \bmod (q - 1)$.
    c. $qInv = q^{-1} \bmod p$.
6. Perform a pair-wise consistency test by verifying that $k = (k^e)^d \bmod n$ for some integer k satisfying $1 < k < n$. If an inconsistency is found, output an indication of a pair-wise consistency failure and stop.

7. Output (*n*, *e*) as the public key, and (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*) as the private key.

Note that key pair validation as specified in Section 6.4.1.2.3 can be performed after step 6 and before step 7. If an error is detected, output an indication of key pair validation failure and stop

6.3.2 RSAKPG2 Family: RSA key pair generation with a random public exponent

The RSAKPG2 family consists of three RSA key pair generators where the public exponent *e* is a random value in the range $65{,}537 \leq e < 2^{256}$.

This family imposes the same constraints on the key pair as in the RSAKPG1 family (see Section 6.3.1).

Three representations are addressed:

1. *rsakpg2-basic* generates the private key in the basic format (*n*, *d*),
2. *rsakpg2-prime-factor* generates the private key in the prime factor format (*p*, *q*, *d*), and
3. *rsakpg2-crt* generates the private key in the Chinese Remainder Theorem format (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*).

An implementation may perform a key pair validation before outputting the key pair from the generation function. The key pair validation methods for this family are specified in Section 6.4.1.3.

6.3.2.1 rsakpg2-basic

rsakpg2-basic is the generator in the RSAKPG2 family where the private key is in the basic format (*n*, *d*).

Function call: *rsakpg2-basic*(*s*, *nBits*, *eBits*)

Input:

1. *s*: the target security strength for the key-pair generator (see Sections 6.2.1 and 6.2.3), an integer in the set {80, 112},
2. *nBits*: the intended length in bits of the RSA modulus, an integer (see Table 1), and
3. *eBits*: the intended length in bits of the public exponent, an integer such that $17 \leq eBits \leq 256$.

Output:

1. (*n*, *e*): the RSA public key, and
2. (*n*, *d*): the RSA private key in the basic format.

Error: Indications of the following:

1. The security strength is out of range,
2. The modulus length is out of range,
3. The exponent length is out of range, or
4. Pair-wise consistency failure.

Process:

1. Check the ranges:
    a. If *s* is not an integer in the set {80, 112}, output an indication that the security strength is out of range and stop.
    b. If *nBits* is not an integer in the set {1024, 2048}, or if *nBits* is less than the minimum key length for the target security strength *s* (see Section 6.2.3), output an indication that the modulus length is out of range and stop.
    c. If *eBits* is not an integer such that $17 \le eBits \le 256$, output an indication that the exponent length is out of range and stop.
2. Generate an odd public exponent *e* in the range $[2^{eBits-1} + 1, 2^{eBits} - 1]$ using an **approved** RBG (see Section 5.3).
3. Generate the prime factors *p* and *q* (see Section 5,4).
4. Determine the private exponent *d*:

$$d = e^{-1} \bmod \text{LCM}(p - 1, q - 1) .$$

5. In the very rare event that $d \le 2^{nBits/2}$, discard *d*, and repeat the process, starting at either step 2 or step 3 (That is, a different value of *e* may be used when generating a new pair of primes, but this is not required).
6. Determine the modulus *n* as $n = p \cdot q$.
7. Perform a pair-wise consistency test by verifying that $k = (k^e)^d \bmod n$ for some integer *k* satisfying $1 < k < n$. If an inconsistency is found, output an indication of a pair-wise consistency failure and stop.
8. Output (*n*, *e*) as the public key, and (*n*, *d*) as the private key.

Note that key pair validation as specified in Section 6.4.1.3.1 can be performed after step 7 and before step 8. If an error is detected, output an indication of key pair validation failure and stop.

6.3.2.2 rsakpg2-prime-factor

rsakpg2-prime-factor is the generator in the RSAKPG2 family where the private key is in the prime factor format (*p*, *q*, *d*).

Function call: *rsakpg2-prime-factor*(*s*, *nBits*, *eBits*)

The inputs, outputs and errors are the same as in *rsakpg2-basic* (see 6.3.2.1), except that the private key is in the prime factor format:

(*p*, *q*, *d*): RSA private key in prime factor format

The steps are the same as in *rsakpg2-basic* except that processing Step 8 is replaced by the following:

8. Output (*n*, *e*) as the public key, and (*p*, *q*, *d*) as the private key.

Note that key pair validation as specified in Section 6.4.1.3.2 can be performed after step 7 and before step 8. If an error is detected, output an indication of key pair validation failure and stop.

6.3.2.3 rsakpg2-crt

rsakpg2-crt is the generator in the RSAKPG2 family where the private key is in the Chinese Remainder Theorem format (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*).

Function call: *rsakpg2-crt*(*s*, *nBits*, *eBits*)

The inputs, outputs and errors are the same as in *rsakpg2-basic* (see 6.3.2.1), except that the private key is in the Chinese Remainder Theorem format:

(*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*): RSA private key in Chinese Remainder Theorem format.

The steps are the same as in *rsakpg2-basic* except that processing Steps 7 and 8 are replaced by the following:

7. Determine the components *dP*, *dQ* and *qInv*:

   a. $dP = d \bmod (p - 1)$.
   b. $dQ = d \bmod (q - 1)$.
   c. $qInv = q^{-1} \bmod p$.

8. Perform a pair-wise consistency test by verifying that $k = (k^e)^d \bmod n$ for some integer k satisfying $1 < k < n$. If an inconsistency is found, output an indication of a pair-wise consistency failure and stop.
9. Output (*n*, *e*) as the public key, and (*n*, *e*, *d*, *p*, *q*, *dP*, *dQ*, *qInv*) as the private key.

Note that key pair validation as specified in Section 6.4.1.3.3 can be performed after step 8 and before step 9. If an error is detected, output an indication of key pair validation failure and stop.

6.4 Assurances of Validity

Secure key establishment depends on the validity of the keys. To explain the assurance requirements, some terminology needs to be defined. The owner of a key pair is the entity that is authorized to use the private key that corresponds to the owner's public key, whether or not the owner generated the key pair. The recipient of a public key is the entity that is participating in a key establishment transaction with the owner and obtains the owner's public key before or during the current transaction.

6.4.1 Assurance of Key Pair Validity

Assurance of key pair validity provides assurance that a key pair was generated in accordance with the requirements of Section 6.2 and Section 6.3. Key pair validity implies public-key validity and assurance of possession of the correct private key. Assurance of key pair validity can only be provided by an entity that has the private key (e.g., the owner). The owner **shall** have assurance of key pair validity before using the key pair for other operations.

6.4.1.1 General Method for Obtaining Assurance of Key Pair Validity

Assurance of key pair validity **shall** be obtained by its owner using (all of) the following steps.

1. Key pair generation: Assurance that the key pair has been correctly formed, in a manner consistent with the criteria of Section 6.2, is obtained using one of the following two methods:

    a. Owner generation – The owner receives the desired assurance if it generates the public/private key pair as specified in Section 6.3.

    b. TTP generation – The owner receives the desired assurance when a trusted third party (TTP) that is trusted by the owner generates the public/private key pair as specified in Section 6.3 and provides it to the owner.

2. The owner **shall** perform a pair-wise consistency test by verifying that $k = (k^e)^d \bmod n$ for some integer k satisfying $1 < k < n$. Note that if the owner generated the key pair (see step 1.a above), an initial pair-wise consistency test was performed during key generation (see Section 6.3). Otherwise, the owner **shall** perform the consistency check separately, prior to the first use of the key pair in a key establishment transaction (see Section 4.1). Additional pair-wise consistency tests **shall** be performed by the owner whenever assurance of key pair validity needs to be refreshed.

3. Key pair validation: A key pair **shall** be validated using one of the following two methods:

    a. Owner key pair validation – The owner either performs a successful key pair validation during key pair generation (see Section 6.3), or performs a successful key pair validation separate from key pair generation (see Sections 6.4.1.2 and 6.4.1.3).

b. TTP key pair validation – A trusted third party (trusted by the owner) either performs a successful key pair validation during key pair generation (see Section 6.3), or performs a successful key pair validation separate from key pair generation (see Sections 6.4.1.2 and 6.4.1.3), and indicates the success to the owner. Note that if the key pair validation is performed separately from the key pair generation, and the TTP does not have the key pair, then the party that generated the key pair or owns the key pair must provide it to the TTP.

A key pair validation **shall** be performed prior to the first use of the key pair in a key establishment transaction (see Section 4.1). The key pair can be revalidated at any time. Note that the use of a TTP to generate a key pair or to perform key pair validation for an owner means that the TTP is trusted (by both the owner and any recipient) to not use the owner's private key to masquerade as the owner or otherwise compromise the key establishment transaction.

6.4.1.2 RSAKPV1 Family: RSA Key Pair Validation with a Fixed Exponent

The RSAKPV1 family of key pair validation methods corresponds to the RSAKPG1 family (see Section 6.3.1).

6.4.1.2.1 *rsakpv1-basic*

rsakpv1-basic is the validation method corresponding to *rsakpg1-basic* (see Section 6.3.1.1).

Function call: *rsakpv1-basic* (*s*, *nBits*, e_{fixed}, (n_{pub}, e_{pub}) (n_{priv}, *d*))

Input:

1. *s*: the target security strength for the key-pair generator (see Section 6.2.3), an integer in the set {80, 112},
2. *nBits*: the expected length in bits of the RSA modulus, an integer (see Table 1),
3. e_{fixed}: the intended fixed public exponent, an odd integer such that $65{,}537 \leq e_{fixed} < 2^{256}$,
4. (n_{pub}, e_{pub}): the RSA public key to be validated, and
5. (n_{priv}, *d*): the RSA private key to be validated in the basic format.

Output:

1. *status*: An indication that the key pair is valid or an indication of an error:
   a. The security strength is out of range,
   b. The modulus length is out of range,
   c. The fixed exponent is out of range, or
   d. The request is invalid.

Process:

1. Check the ranges:
    a. If s is not an integer in the set {80, 112}, output an indication that the security strength is out of range and stop.
    b. If *nBits* is not an integer in the set {1024, 2048}, or if *nBits* is less than the minimum key length for the target security strength s (see [7]), output an indication that the modulus length is out of range and stop.
    c. If e_{fixed} is not an odd integer such that $65{,}537 \leq e_{fixed} < 2^{256}$, output an indication that the fixed exponent is out of range and stop.
2. Compare the public exponents:

    If $e_{\text{pub}} \neq e_{\text{fixed}}$, output an indication that the request is invalid and stop.
3. Check the modulus:
    a. If $n_{\text{pub}} \neq n_{\text{priv}}$, output an indication of an invalid key pair and stop.
    b. If the length in bits of the modulus n_{pub} is not *nBits*, output an indication of an invalid key pair and stop.
4. Prime factor recovery:
    a. Recover the prime factors p and q from the modulus n_{pub}, the public exponent e_{pub} and the private exponent d (see Appendix C):

        (p, q) = RecoverPrimeFactors (n_{pub}, e_{pub}, d)
    b. If RecoverPrimeFactors outputs an indication that the prime factors were not found, output an indication that the request is invalid and stop.
    c. If $n_{\text{pub}} \neq p \cdot q$, then output an indication that the request is invalid and stop.
5. Check the prime factors:
    a. Apply an **approved** primality test to test the prime number p (see Section 5.5).
    b. If the primality test indicates that p is not prime, output an invalid key pair and stop.
    c. If $(p < \sqrt{2}(2^{\,nBits/2 - 1}))$ or $(p > 2^{\,nBits/2} - 1)$, output an indication of an invalid key pair and stop.
    d. If GCD $(p - 1, e_{pub}) \neq 1$, output an indication of an invalid key pair and stop.
    e. Apply an **approved** primality test to test the prime number q (see Section 5.5).
    f. If the primality test indicates that q is not prime, output an indication of an invalid key pair and stop.

g. If ($q < \sqrt{2}(2^{nBits/2-1})$) or ($q > 2^{nBits/2} - 1$), output an indication of an invalid key pair and stop.

h. If GCD ($q - 1$, e_{pub}) $\neq 1$, output an indication of an invalid key pair and stop.

i. If $|p - q| \leq 2^{nBits/2-100}$, output an indication of an invalid key pair and stop.

6. Check that the private exponent d satisfies

   a. $2^{nBits/2} < d <$ LCM ($p - 1$, $q - 1$).

   and

   b. $1 = (d\ e_{pub})$ mod LCM ($p - 1$, $q - 1$).

   If either check fails, output an indication of an invalid key pair and stop.

7. Output an indication that the key pair is valid.

6.4.1.2.2 *rsakpv1-prime-factor*

rsakpv1-prime-factor is the validation method corresponding to *rsakpg1-prime-factor* (see 6.3.1.2).

Function call: *rsakpv1-prime-factor* (s, *nBits*, e_{fixed}, (n_{pub}, e_{pub}), (p, q, d))

The inputs, outputs and errors are the same as in *rsakpv1-basic* (see Section 6.4.1.2.1), except that the private key is in the prime factor format:

(p, q, d)

The steps are the same as in *rsakpv1-basic* except that in processing:

1. Step 3 is replaced by the following:

   3. Check the modulus:

      a. If $n_{pub} \neq p \cdot q$, output an indication of an invalid key pair and stop.

      b. If the length in bits of the modulus n_{pub} is not *nBits*, output an indication of an invalid key pair and stop.

2. Step 4 (prime factor recovery) is omitted.

6.4.1.2.3 *rsakpv1-crt*

rsakpv1-crt is the validation method corresponding to *rsakpg1-crt*.

Function call: *rsakpv1-crt* (s, *nBits*, e_{fixed}, (n_{pub}, e_{pub}), (n_{priv}, e_{priv}, d, p, q, dP, dQ, $qInv$))

The inputs, outputs and errors are the same as in *rsakpv1-basic* (see Section 6.4.1.2.1), except that the private key is in the Chinese Remainder Theorem format:

(n_{priv}, e_{priv}, d, p, q, dP, dQ, $qInv$)

The steps are the same as in *rsakpv1-basic* except that in processing:

1. Step 2 is replaced by the following:

   2. Compare the public exponents:

      If $e_{pub} \neq e_{fixed}$ or $e_{pub} \neq e_{priv}$, output an indication of an invalid key pair and stop.

2. Step 3 is replaced by

   3. Check the modulus:

      a. If $n_{pub} \neq p \cdot q$, or $n_{pub} \neq n_{priv}$, output an indication of an invalid key pair and stop.

      b. If the length in bits of the modulus n_{pub} is not *nBits*, output an indication of an invalid key pair and stop.

3. Step 4 (prime factor recovery) is omitted,

4. Step 7 is replaced by the following:

   7. Check the CRT components: Check that the components *dP*, *dQ* and *qInv* satisfy

      a. $1 < dP < (p - 1)$.

      b. $1 < dQ < (q - 1)$.

      c. $1 < qInv < p$.

      d. $(dP \cdot e_{fixed}) - 1 = 0 \bmod (p - 1)$.

      e. $(dQ \cdot e_{fixed}) - 1 = 0 \bmod (q - 1)$.

      f. $(qInv \cdot q) - 1 = 0 \bmod p$.

      If any of the criteria are not met, output an indication of an invalid key pair and stop.

   8. Output an indication that the key pair is valid.

6.4.1.3 RSAKPV2 Family: RSA Key Pair Validation with a Random Exponent

The RSAKPV2 family of key pair validation methods corresponds to RSAKPG2 family (see Section 6.3.2).

6.4.1.3.1 *rsakpv2-basic*

rsakpv2-basic is the validation method corresponding to *rsakpg2-basic* (see Section 6.3.2.1).

Function call: *rsapkv2-basic* (*s*, *nBits*, *eBits*, (n_{pub}, *e*), (n_{priv}, *d*))

The method is the same as the *rsapkv1-basic* method in Section 6.4.1.2 except that:

1. The e_{fixed} input parameter becomes *eBits*, which is the expected length in bits of the public exponent, an integer such that $17 \leq eBits \leq 2^{256}$.
2. Step 1c is replaced by:

   c. If (*eBits* < 17) or (*eBits* > 256), output an indication that the exponent is out of range and stop.
3. Step 2 is replaced by:

   2. Check the public exponent.

      If the public exponent e_{pub} is not odd, or if the length in bits of the public exponent e_{pub} is not *eBits*, output an indication of an invalid key pair and stop.

6.4.1.3.2 *rsakpv2-prime-factor*

rsakpv2-prime-factor is the validation method corresponding to *rsakpg2-prime-factor* (see Section 6.3.2.2).

Function call: *rsakpv2-prime-factor* (*s*, *nBits*, *eBits*, (n_{pub}, e_{pub}), (*p*, *q*, *d*))

The inputs, outputs and errors are the same as in *rsakpv2-basic* (see Section 6.4.1.3.1), except that the private key is in the prime factor format:

(*p*, *q*, *d*)

The steps are the same as in *rsakpv2-basic* except that in processing:

1. Step 3 is replaced by the following:

   3. Check the modulus:

      a. If $n_{pub} \neq p \cdot q$, output an indication of an invalid key pair and stop.

      b. If the length in bits of the modulus n_{pub} is not *nBits*, output an indication of an invalid key pair and stop.
2. Step 4 (prime factor recovery) is omitted.

6.4.1.3.3 *rsakpv2-crt*

rsakpv2-crt is the validation method corresponding to *rsakpg2-crt* (see Section 6.3.1.3).

Function call: *rsakpv2-crt* (*s*, *nBits*, *eBits*, (n_{pub}, e_{pub}), (n_{priv}, e_{priv}, *d*, *p*, *q*, *dP*, *dQ*, *qInv*))

The inputs, outputs and errors are the same as in *rsakpv2-basic* (see Section 6.4.1.3.1), except that the private key is in the Chinese Remainder Theorem format:

(n_{priv}, e_{priv}, *d*, *p*, *q*, *dP*, *dQ*, *qInv*)

The steps are the same as in *rsakpv2-basic* except that in processing:

1. Step 2 is replaced by the following:

   2. Compare the public exponents:

      If ($e_{pub} \neq e_{priv}$) or (e_{pub} is not odd) or (e_{pub} is not *eBits*), output an indication of an invalid key pair and stop.

2. Step 3 is replaced by

   3. Check the modulus:

      a. If ($n_{pub} \neq pq$) or ($n_{pub} \neq n_{priv}$) output an indication of an invalid key pair and stop.

      b. If the length in bits of the modulus n_{pub} is not *nBits*, output an indication of an invalid key pair and stop.

3. Step 4 (prime factor recovery) is omitted,

4. Step 7 is replaced by the following:

   7. Check the CRT components: Check that the components *dP*, *dQ* and *qInv* satisfy

      a. $1 < dP < (p - 1)$.

      b. $1 < dQ < (q - 1)$.

      c. $1 < qInv < p$.

      d. $1 = (dP \cdot e_{pub}) \bmod (p - 1)$.

      e. $1 = (dQ \cdot e_{pub}) \bmod (q - 1)$.

      f. $1 = (qInv \cdot q) \bmod p$.

      If any of the criteria are not met, output an indication of an invalid key pair and stop.

   8. Output an indication that the key pair is valid.

6.4.2 Recipient Assurances of Public Key Validity

In this Recommendation, the recipient of a public key is an entity that does not (and should not) have access to the associated private key. The recipient of a candidate public key **shall** have assurance of the arithmetic validity of that key before using it in a key establishment transaction with its owner.

6.4.2.1 General Method for Obtaining Assurance of Public Key Validity

The recipient **shall** obtain assurance of public key validity using one or more of the following methods:

1. Recipient Partial Public Key Validation - The recipient performs a successful partial public key validation (see Section 6.4.2.2).
2. TTP Partial Public Key Validation – The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful partial public key validation (see Section 6.4.2.2).
3. TTP Key Pair Validation – The recipient receives assurance that a trusted third party (trusted by the recipient and the owner) has performed key pair validation in accordance with Section 6.4.1.1 (step 3.b).

Note that the use of a TTP to perform key pair validation (method 3) implies that both the owner and any recipient of the public key trust that the TTP will not use the owner's private key to masquerade as the owner or otherwise compromise the key establishment transaction.

6.4.2.2 Partial Public Key Validation for RSA

Partial public key validation for RSA consists of conducting plausibility tests. These tests determine whether the public modulus and public exponent are plausible, not necessarily whether they are completely valid, i.e., they may not conform to all RSA key generation requirements as specified in this Recommendation. Plausibility tests can detect unintentional errors with a reasonable probability. Note that full RSA public key validation is not specified in this Recommendation, as it is an area of research. Therefore, if an application requires assurance of full public key validation, then another **approved** key establishment method **shall** be used.

Plausibility tests **shall** include the tests specified in SP 800-89, Section 5.3.3 with the caveat that the length of the modulus **shall** be a length that is specified in this Recommendation.

6.5 Assurances of Private Key Possession

The security of key agreement schemes that use key pairs depends on limiting knowledge of the private keys to those who have been authorized to access them (e.g., their respective owners or certain trusted third parties). In addition to preventing unauthorized entities from gaining access to private keys, it is also important to obtain assurance that authorized owners actually have access to their correct private keys.

Assurance of possession requirements for the owner of a private key are specified in Section 6.5.1. Parties that interact with the owner (e.g., a public key recipient) also need to obtain assurance that the owner possesses the private key; these requirements are specified in Section 6.5.2.

When assurance of possession of a private key is initially obtained, the assurance of the validity of the associated public key **shall** be obtained either prior to or concurrently with obtaining assurance of possession. Note that as time passes, an owner could lose possession of the associated private key, deliberately or due to an error; for this reason, renewing the assurance of possession may be appropriate for some applications (i.e., assurance of possession can be refreshed). See Section 6.5.2.2 and Section 6.6 for a discussion of the methods that may be used by the recipient of a public key to renew the assurance of the owner's possession of the

corresponding private key. A discussion of the effect of time on the assurance of private key possession is provided in SP 800-89.

6.5.1 Owner Assurance of Private Key Possession

The owner of a public key **shall** have assurance that the owner actually possesses the correct associated private key in one or more of the following ways:

1. Owner Receives Assurance via Key Generation - The act of generating a key pair, as specified in this document, is a way for the owner to obtain assurance of possession of the correct private key. This method allows an owner who protects his/her own keys to have assurance of possession without additional computation.

2. Owner Receives Assurance via Assurance of Key Pair Validity – The owner obtains assurance of key pair validity (See section 6.4.1) thereby also obtaining owner assurance of private key possession.

3. Owner Receives Assurance via Explicit Key Confirmation – The owner employs the key pair to successfully engage another party in a key agreement transaction using a scheme from the KAS2 family that incorporates explicit key confirmation. The key confirmation **shall** be performed with the owner as key confirmation recipient in order to obtain assurance that the private key functions correctly. See Section 6.6 for further explanation.

4. Owner Receives Assurance via an Encrypted Certificate - The owner uses the private key while engaging in a key establishment transaction with a Certificate Authority (trusted by the owner), after providing the CA with the corresponding public key. As part of this transaction, the CA generates a (new) certificate containing the owner's public key and encrypts that certificate using (some portion of) the symmetric keying material that has been established. Only the encrypted form of the certificate is provided to the owner. By successfully decrypting the certificate and verifying the CA's signature, the owner obtains assurance of possession of the correct private key (at the time of the key establishment transaction).

The owner of a public key (or agents trusted to act on the owner's behalf) **should** determine that the method used for obtaining assurance of the owner's possession of the correct private key is sufficient and appropriate to meet the security requirements of the owner's intended application(s)

6.5.2 Recipient Assurance of Owner's Possession of a Private Key

It is assumed that, at the time of binding an identifier to the owner's public key, the binding authority has obtained assurance that the owner is in possession of the correct private key. In conjunction with a (successful) key establishment transaction, the recipient of another party's public key **shall** also obtain this assurance – either indirectly using a trusted third party (see Section 6.5.2.1) or directly from the claimed owner (see Section 6.5.2.2) - before using the established keying material for purposes beyond those required during the key establishment transaction itself.

When two parties engage in a key establishment transaction, there is (at least) an implicit claim of ownership made whenever a public key is provided on behalf of a particular party. That party is considered to be a *claimed* owner of the corresponding key pair – as opposed to being a *true* owner – until adequate assurance can be provided that the party is actually the one in possession of the private key (See Section 6.7).

The recipient of a public key (or agents trusted to act on the recipient's behalf) **should** determine that the method used for obtaining assurance of the owner's possession of the correct private key is sufficient and appropriate to meet the security requirements of the owner's intended application(s).

6.5.2.1 Recipient Indirectly Obtains Assurance of Possession Using a Trusted Third Party

The recipient of a public key may indirectly receive assurance that its owner is in possession of the correct private key using a trusted third party, either before or during a key establishment transaction that makes use of that public key. The methods used by a third party trusted by the recipient to obtain that assurance are beyond the scope of this Recommendation (see, however, Section 6.5.2.2 of this Recommendation and Section 8.1.5.1.1.2 of SP 800-57 Part 1 [7] for possible methods). The recipient of a public key (or agents trusted to act on behalf of the recipient) **should** know the method(s) used by the third party, in order to determine that the assurance obtained on behalf of the recipient is sufficient and appropriate to meet the security requirements of the recipient's intended application(s).

6.5.2.2 Recipient Obtains Assurance of Possession Directly from the Claimed Owner

The recipient of a public key can directly obtain assurance of the claimed owner's current possession of the corresponding private key by successfully completing a key establishment transaction that explicitly incorporates key confirmation as specified in Sections 8.2.3, 8.3.3, 9.2.4, or 9.3.4 with the claimed owner serving as the key confirmation provider. Note that the recipient of the public key in question will also be the key confirmation recipient. (See Section 6.6 for further explanation.) Also note that this use of key confirmation is an additional benefit beyond its use to confirm that two parties possess the same keying material.

The recipient of a public key (or agents trusted to act on the recipient's behalf) **shall** determine whether or not using one of the key establishment schemes in this Recommendation to obtain assurance of possession through key confirmation is sufficient and appropriate to meet the security requirements of the recipient's intended application(s). Other **approved** methods (e.g. Section 5.4.4 of SP800-57 Part 1) of directly obtaining this assurance of possession from the owner are also allowed. If obtaining assurance of possession directly from the owner is not acceptable, then assurance of possession **shall** be obtained indirectly as discussed in Section 6.5.2.1.

Successful key confirmation (performed in the context described in this Recommendation) demonstrates that the correct private key has been used in the key confirmation provider's calculations, and thus also provides assurance that the claimed owner is the true owner

The assurance of possession may be useful even when the recipient has previously obtained independent assurance that the claimed owner of a public key is indeed its true owner. This may be appropriate in situations where the recipient desires renewed assurance that the owner possesses the correct private key (and that the owner is still able to use it correctly), including situations where there is no access to a trusted party who can provide renewed assurance of the owner's continued possession of the private key.

Note that the requirement that assurance of possession be obtained before using the established keying material for purposes *beyond* those of the key establishment transaction itself does not prohibit the parties to a key establishment transaction from using a portion of the derived or transported keying material *during* the key establishment transaction, for purposes required by that key establishment scheme. For example, in a transaction involving a key agreement scheme that incorporates key confirmation, the parties establish a (purported) shared secret, derive keying material, and — as part of that same transaction — use a portion of the derived keying material as the *MacKey* in their key confirmation computations.

6.6 Key Confirmation

The term *key confirmation* (KC) refers to actions taken to provide assurance to one party (the key confirmation *recipient*) that another party (the key confirmation *provider*) is in possession of a (supposedly) shared secret and/or the correct version of keying material that was derived or transported during a key establishment transaction. (Correct from the perspective of the key confirmation recipient.) Such actions are said to provide *unilateral key confirmation* when they provide this assurance to only one of the participants in a key establishment transaction; the actions are said to provide *bilateral key confirmation* when this assurance is provided to both participants (that is, unilateral key confirmation is provided in both directions).

Oftentimes, key confirmation is provided implicitly by some means outside of the key establishment scheme (for example, by decrypting an encrypted message sent from the other party using a symmetric key that was derived, in part, from a "master secret" determined during the key establishment transaction), but this is not always the case. Some schemes in the Recommendation include the exchange of explicit key confirmation information in order to enhance the scheme's security properties.

In this Recommendation, key confirmation can be provided only if the provider owns a key-establishment pair that is used during key establishment. Each family of key agreement schemes specified in this Recommendation includes a scheme that incorporates unilateral key confirmation provided by the responder to the initiator. Similarly, each family of key transport schemes specified in this Recommendation includes a scheme that incorporates unilateral key confirmation provided by the receiver to the sender. The KAS2 family of key agreement schemes also includes a scheme incorporating unilateral key confirmation provided by the initiator to the responder, and a scheme incorporating bilateral key confirmation.

In each scheme that includes key confirmation, the following steps **shall** be performed:

1. The KC recipient sends unpredictable secret data to the KC provider encrypted using the provider's public key to produce (ephemeral) ciphertext.

2. The KC provider uses its private key to decrypt the ciphertext before obtaining (or deriving) a *MacKey* that is randomly generated for each transaction, and known only to the parties engaged in that transaction.

3. This *MacKey* and certain transaction-specific *MacData* (which includes the parties' identifiers as well as ephemeral data that has been exchanged between the parties) are used by the KC provider as input to an **approved** MAC algorithm to obtain a *MacTag* whose value is (for all practical purposes) unique to the transaction. The *MacTag* value is then sent to the KC recipient. (See Sections 5.2, 5.9, 6.2, 8 and 9 for details.

4. The KC recipient performs an independent computation of the *MacTag*. If the *MacTag* value computed by the KC recipient matches the *MacTag* value received from the KC provider, then key confirmation is successful, and the KC recipient obtains assurance that both parties contemporaneously agree on the values of the *MacKey* and *MacData*, and that they have successfully established a shared secret and/or keying material. The *MacKey* employed during the transaction **shall** be zeroized after its use. The *MacKey* **shall not** be used for purposes other than key confirmation or implementation validation testing. (See Sections 5.2, 5.9, 6.2, 8 and 9 for details).

A close examination of the KC process shows that any of the six key establishment schemes specified in this Recommendation that incorporate key confirmation can be used to provide the KC recipient with assurance that the KC provider is currently in possession of the (correct) private key – the one corresponding to the KC provider's public key-establishment key.

The transaction-specific values of both the *MacKey* and *MacData* prevent (for all practical purposes) the replay of any previously computed value of *MacTag*. The receipt of a correctly computed *MacTag*, coupled with the presumed inability of the KC provider (or others) to predict the values of either the *MacKey* or the secret data that was encrypted with the KC provider's public key, provides assurance to the KC recipient that the KC provider has used the correct private key during the current transaction – to successfully recover the secret data that is a prerequisite to learning the value of the *MacKey*.

The three key-agreement schemes in the KAS2 family that incorporate key confirmation can also be used to provide assurance to the KC recipient that it is in possession of the correct value of the private key corresponding to its own public key-agreement key. In each of these schemes, the KC recipient receives an unpredictable secret value from the KC provider that has been encrypted using the KC recipient's public key to form (ephemeral) ciphertext. The KC recipient uses its private key to decrypt this ciphertext, and includes the recovered shared secret as part of the input to the KDF when deriving the *MacKey* described above. (See Section 8.3.) If the KC recipient used an incorrect value for its private key, it is highly likely that it would not recover the correct version of that shared secret, and as a result, would incorrectly compute both the *MacKey* and *MacTag*. Therefore, if the *MacTag* value computed by the KC recipient in a KAS2 scheme matches the *MacTag* value received from the KC provider, then – in addition to the other types of assurance described above – the KC recipient obtains assurance that it has used the correct private key during the current transaction to successfully recover the secret value sent by the KC provider.

In order to assert that key confirmation is performed in compliance with this Recommendation, key confirmation **shall** be incorporated into a key establishment scheme as specified in this Recommendation. If any other methods are used to provide key confirmation, this Recommendation makes no statement as to their adequacy.

6.6.1 Unilateral Key Confirmation for Key Establishment Schemes

As specified in this Recommendation, unilateral key confirmation occurs when one participant in a key establishment scheme (the "provider") successfully provides assurance to the other participant (the "recipient") that both the provider and the recipient have computed the same secret *MacKey* during a key establishment transaction. This *MacKey* is either 1) derived from a shared secret determined during a key agreement transaction, or 2) included in the secret keying material shared by both the provider and the recipient during a key transport transaction. In this Recommendation, the inclusion of key confirmation in a scheme is restricted to cases where the key confirmation provider owns a key-establishment key pair that is used during the establishment transaction.

As a necessary part of the process of providing/obtaining unilateral key confirmation, the following steps **shall** be incorporated into a key establishment scheme. (Depending upon the particular key establishment scheme, additional steps may also be required. Details will be provided for each scheme.) Note that the provider may be either the scheme initiator/sender (party U) or the scheme responder/receiver (party V), as long as the provider is using a key-establishment key pair in the key establishment scheme, and the recipient is the other party.

1. The provider computes

   $MacData_P = message\_string_P \| ID_P \| ID_R \| EphemData_P \| EphemData_R \{\| Text\}$

   where

   $message\_string_P$ is a six byte string with a value of "KC_1_U" or "KC_1_V", depending on whether U or V is providing the *MacTag*. Note that these values will differ for bilateral key confirmation as specified in Section 6.6.2.

   ID_P is the identifier of the provider.

   ID_R is the identifier of the recipient.

   $EphemData_P$ and $EphemData_R$ are ciphertext values or nonces contributed by the provider and recipient, respectively. These values are specified in the sections describing the schemes that include key confirmation. $EphemData_P$ is null in the key transport cases.

   Text is an optional byte string that may be used during key confirmation and that is known by the parties establishing the secret keying material.

2. In the case of a key agreement scheme: After computing the shared secret and applying the key derivation function to obtain the *DerivedKeyingMaterial* (see Section 5.9), the provider parses *DerivedKeyingMaterial* into two parts, the *MacKey* and the *KeyData*:

   $MacKey \| KeyData = DerivedKeyingMaterial.$

In the case of a key transport scheme, the provider parses the *TransportedKeyingMaterial* into the same two parts:

MacKey || *KeyData* = *TransportedKeyingMaterial.*

3. The provider computes $MacTag_P$ (see Section 5.2.1) and sends it to the recipient:

   $MacTag_P$ = MAC(*MacKey*, *MacTagLen*, $MacData_P$).

4. The recipient determines $MacData_P$, *MacKey*, and $MacTag_P$ in the same manner as the provider, and then compares its computed $MacTag_P$ to the value received from the provider. If *they are equal*, then the recipient is assured that the provider has used the same value for *MacKey* in its computations and that the provider shares the recipient's value of $MacData_P$. The assurance of a shared value for *MacKey* provides additional assurance to the recipient that:

   a. For key agreement schemes: the provider shares the secret value (*Z*) from which *MacKey* and *KeyData* are derived (see Section 5.9). Thus, the recipient also has assurance that the provider could compute *KeyData* correctly.

   b. For key transport schemes: The provider has recovered all of the transported keying material including *KeyData.*

5. Zeroize the *MacKey* once it is no longer needed for *MacTag* computations.

6.6.2 Bilateral Key Confirmation for Key Establishment Schemes

Bilateral key confirmation is accomplished by performing unilateral key confirmation in both directions (with U providing $MacTag_U$ to recipient V, and V providing $MacTag_V$ to recipient U) during the same scheme. In addition to replacing P and R by U and V (or by V and U), there are also a few clarifications to the process described in Section 6.6.1:

1. When computing $MacTag_U$, the value of the six-byte $message\_string_U$ that forms the initial segment of $MacData_U$ is "KC_2_U".

2. When computing $MacTag_V$, the value of the six-byte $message\_string_V$ that forms the initial segment of $MacData_V$ is "KC_2_V".

3. If used at all, the value of the (optional) byte string *Text* used to form the final segment of $MacData_U$ can be different than the value of *Text* used to form the final segment of $MacData_V$.

6.7 Authentication

Successful key confirmation, when performed as specified in this Recommendation, can supply entity authentication with respect to the key confirmation provider; i.e., the key confirmation recipient can obtain assurance concerning the identity of the provider.

The correct computation of $MacTag_P$ by the provider requires knowledge of the private key corresponding to a particular public key-establishment key that has been bound to the key-pair

owner's identifier. The recipient of a correctly computed $MacTag_P$ (correct from the recipient's perspective) obtains assurance that the provider is in possession of the correct private key, and may infer that the provider is the owner of that key pair.

In addition to the security level associated with the cryptographic elements and parameters employed during the key-establishment/key-confirmation process, the level of assurance associated with this entity-authentication technique is dependent upon the specificity of the key-pair owner's identifier.

7 IFC Primitives and Operations

7.1 Encryption and Decryption Primitives

RSAEP and RSADP are the basic encryption and decryption primitives from the RSA cryptosystem [16]. RSAEP produces ciphertext from keying material using a public key; RSADP recovers the keying material from the ciphertext using the corresponding private key.

7.1.1 RSAEP

RSAEP produces ciphertext from keying material using an RSA public key.

Function call: RSAEP((*n*, *e*), *k*)

Input:

1. (*n*, *e*): the RSA public key.
2. *k*: the keying material, an integer such that $1 < k < n - 1$.

Output:

c: the ciphertext, an integer such that $1 < c < n - 1$.

Errors: An indication that the keying material is out of range.

Assumption: The RSA public key is valid (see Section 6.4).

Process:

1. If the keying material *k* is not such that $1 < k < n - 1$, output an indication that the keying material is out of range and stop.
2. Let $c = (k)^e \bmod n$.
3. Output *c*.

7.1.2 RSADP

RSADP is the basic decryption operation. It recovers keying material from ciphertext using an RSA private key.

Function call: RSADP((*n, d*), *c*)

Input:

1. (*n, d*): the RSA private key.
2. *c*: the ciphertext, such that $1 < c < n - 1$.

Output:

k: the keying material, an integer such that $1 < k < n - 1$.

Errors: An indication that the ciphertext is out of range.

Assumption: The RSA private key is part of a valid key pair (see Section 6.4).

Process:

1. If the ciphertext *c* is not such that $1 < c < n - 1$, output an indication that the ciphertext is out of range and stop.
2. Let $k = c^d \bmod n$.
3. Output *k*.

Note:

Care needs to be taken to ensure that an implementation of RSADP does not reveal even partial information about the value of *k*. An opponent who can reliably obtain particular bits of *k* for sufficiently many chosen ciphertexts may be able to obtain the full decryption of an arbitrary ciphertext by applying the bit-security results of Håstad and Näslund [17].

7.2 Encryption and Decryption Operations

7.2.1 RSA Secret Value Encapsulation (RSASVE)

Secret value encapsulation generates and encrypts a secret value to produce ciphertext using a public key-establishment key. The recovery operation recovers the secret value (now the shared secret) from the ciphertext using the corresponding private key-establishment key. Secret value encapsulation employs a Random Bit Generator (RGB) to generate the secret value.

The RSASVE generate and recovery operations specified in Sections 7.2.1.2 and 7.2.1.3, respectively, are based on the RSAEP and RSADP primitives (see Section 7.1). These operations are used by the KAS1 and KAS2 key agreement families (see Sections 8.2 and 8.3), and by the RSA-KEM KWS key transport family (see Sections 9.3 and 7.2.3).

7.2.1.1 RSASVE Components

RSASVE requires an **approved** RBG (see Section 5.3). The security strength for the RBG **shall** be equal to or greater than the target security strength for the schemes in which RSASVE is employed.

1. RSAEP: RSA Encryption Primitive (see Section 7.1.1).
2. RSADP: RSA Decryption Primitive (see Section 7.1.2).

7.2.1.2 RSASVE Generate Operation

RSASVE.GENERATE generates a shared secret and corresponding ciphertext using an RSA public key.

Function call: **RSASVE.GENERATE**((*n*, *e*))

Input:

(*n*, *e*): an RSA public key.

Output:

1. *Z*: the shared secret; a byte string of length *nLen* bytes.
2. *C*: the ciphertext; a byte string of length *nLen* bytes.

Assumptions: The RSA public key is part of a valid key pair.

Process:

1. Compute the value of *nLen* as the length in bytes of the modulus *n*.
2. Generation:
    a. Using the RBG (see Section 5.3), generate an *nLen* byte string, *Z*.
    b. Convert *Z* to an integer *z* (See Appendix B.2):

        $z = \text{BS2I}(Z, nLen)$.

    c. If z does not satisfy $1 < z < n - 1$, then go to a.
3. RSA encryption:
    a. Apply the RSAEP encryption primitive (see Section 7.1.1) to the integer *z* using the public key (*n*, *e*) to produce an integer ciphertext *c*:

        $c = \text{RSAEP}((n, e), z)$.

    b. Convert the ciphertext *c* to a ciphertext byte string *C* of length *nLen* bytes (see Appendix B.1):

        $C = \text{I2BS}(c, nLen)$.

4. Output the string *Z* as the shared secret, and the ciphertext *C*.

7.2.1.3 RSASVE Recovery Operation

RSASVE.RECOVER recovers a shared secret from ciphertext using an RSA private key.

Function call: RSASVE.RECOVER((*n*, *d*), *C*)

Input:

1. (*n*, *d*): an RSA private key.
2. *C*: the ciphertext; a byte string of length *nLen* bytes.

Output:

Z: the shared secret; a byte string of length *nLen* bytes.

Errors: An indication of a decryption error.

Assumptions: The RSA private key is part of a valid key pair.

Process:

1. Compute the value of *nLen* as the length in bytes of the modulus *n*.
2. Length checking:

   If the length of the ciphertext *C* is not *nLen* bytes, output an indication of a decryption error and stop.

3. RSA decryption:

   a. Convert the ciphertext *C* to an integer ciphertext *c* (see Appendix B.2):

   $$c = \text{BS2I}(C).$$

   b. Apply the RSADP decryption primitive (see Section 7.1.2) to the ciphertext *c* using the private key (*n, d*) to produce an integer *z*:

   $$z = \text{RSADP}((n, d), c) .$$

   c. If RSADP indicates that the ciphertext is out of range, output an indication of a decryption error and stop.

   d. Convert the integer *z* to a byte string *Z* of length *nLen* bytes (see Appendix B.1):

   $$Z = \text{I2BS}(z, nLen).$$

4. Output the string *Z* as the shared secret.

Note:

Care needs to be taken to ensure that an implementation does not reveal information about the encapsulated secret value *Z*. For instance, the observable behavior of the I2BS routine must not reveal even partial information about the byte string *Z*. An opponent who can reliably obtain particular bits of *Z* for sufficiently many chosen ciphertexts may be able to obtain the full decryption of an arbitrary RSA-encrypted value by applying the bit-security results of Håstad and Näslund [17].

7.2.2 RSA with Optimal Asymmetric Encryption Padding (RSA-OAEP)

RSA-OAEP consists of asymmetric encryption and decryption operations that are based on an **approved** hash function, an **approved** random bit generator, a mask generation function, and the RSAEP and RSADP primitives. These operations are used by the KTS-OAEP key transport schemes (see Section 9.2).

In the RSA-OAEP encryption operation, a data block is constructed from the keying material to be transported and the hash of additional input (see Section 9.1) that is shared by the sender and the intended receiving party. A random byte string is generated, after which both the random byte string and the data block are masked in a way that binds their values. The masked values are used to form the plaintext that is input to the RSAEP primitive, along with the public key-establishment key of the intended receiving party. The resulting RSAEP output further binds the random byte string, the keying material and the hash of the additional data in the ciphertext that is sent to the receiving party.

In the RSA-OAEP decryption operation, the ciphertext and the receiving party's private key-establishment key are input to the RSADP primitive, recovering the masked values as output. The mask generating function is then used to reconstruct and remove the masks that obscure the random byte string and the data block. After removing the masks, the receiving party can examine the format of the recovered data, and can compare its own computation of the hash of the additional data to the hash value contained in the unmasked data block, thus obtaining some measure of assurance of the integrity of the recovered data – including the transported keying material.

RSA-OAEP can process up to $nLen - 2(hLen) - 2$ bytes of keying material, where *nLen* is the length of the recipient's RSA modulus, and *hLen* is the length (in bytes) of the values output by the underlying hash function.

7.2.2.1 RSA-OAEP Components

RSA-OAEP uses the following components:

1. H: An **approved** hash function (see Section 5.1). *hLen* is used to denote the length (in bytes) of the hash function output.

2. MGF: The mask generation function (see Section 5.8).

3. RBG: An **approved** random bit generator (see Section 5.3).
4. RSAEP: RSA Encryption Primitive (see Section 7.1.1).
5. RSADP: RSA Decryption Primitive (see Section 7.1.2).

The security strength for the RBG **shall** be at least the target security strength for the scheme.

7.2.2.2 RSA-OAEP Encryption Operation

The RSA-OAEP.Encrypt operation produces a ciphertext from keying material and additional input using an RSA public key as shown in Figure 4. See Section 9.1 for more information on the additional input. Let *hLen* be the length of the hash function output in bytes.

Function call: RSA-OAEP.ENCRYPT((*n*, *e*), *K*, *A*)

Input*:*

1. (*n*, *e*): the receiver's RSA public key.
2. *K*: the keying material; a byte string of length at most $nLen - 2hLen - 2$.
3. *A*: additional input; a byte string (may be the empty string) to be cryptographically bound to the keying material (see Section 9.1).

Output:

C: the ciphertext; a byte string of length *nLen* bytes.

Errors: An indication that the keying material is too long.

Assumptions: The RSA public key is valid.

Process*:*

1. *nLen* = the length of *n* in bytes.
2. Length checking:
    a. *KLen* = the length of *K* in bytes.
    b. If $KLen > nLen - 2hLen - 2$, then output an indication that the keying material is too long and stop.
3. OAEP encoding:
    a. Apply the selected hash function to compute:

$$HA = \mathrm{H}(A).$$

HA is a byte string of length *hLen*. If *A* is an empty string, then *HA* is the hash value for the empty string.

b. Construct a byte string *PS* consisting of *nLen* – *KLen* – 2*hLen* – 2 zero bytes. The length of *PS* may be zero.

c. Concatenate *HA*, *PS*, a single byte with hexadecimal value of 01, and the keying material *K* to form data *DB* of length *nLen* – *hLen* – 1 bytes as follows:

$$DB = HA \,||\, PS \,||\, 01 \,||\, K,$$

where 01 represents a byte.

d. Using the RBG (see Section 5.3), generate a random byte string *mgfSeed* of length *hLen* bytes.

e. Apply the mask generation function in Section 5.8 to compute:

$$dbMask = \text{MGF}(mgfSeed, nLen - hLen - 1).$$

f. Let $maskedDB = DB \oplus dbMask$.

g. Apply the mask generation function in Section 5.8 to compute:

$$mgfSeedMask = \text{MGF}(maskedDB, hLen).$$

h. Let $maskedMGFSeed = mgfSeed \oplus mgfSeedMask$.

i. Concatenate a single byte with hexadecimal value 00, *maskedMGFSeed*, and *maskedDB* to form an encoded message *EM* of length *nLen* bytes as follows:

$$EM = 00 \,||\, maskedMGFSeed \,||\, maskedDB$$

where 00 represents a byte.

4. RSA encryption:

a. Convert the encoded message *EM* to an integer *em* (see Appendix B.2):

$$em = \text{BS2I}(EM).$$

b. Apply the RSAEP encryption primitive (see Section 7.1.1) to the integer *em* using the public key (*n*, *e*) to produce a ciphertext integer *c*:

$$c = \text{RSAEP}((n, e), em).$$

c. Convert the ciphertext integer *c* to a ciphertext byte string *C* of length *nLen* bytes (see Appendix B.1):

$$C = \text{I2BS}(c, nLen).$$

5. Output the ciphertext *C*.

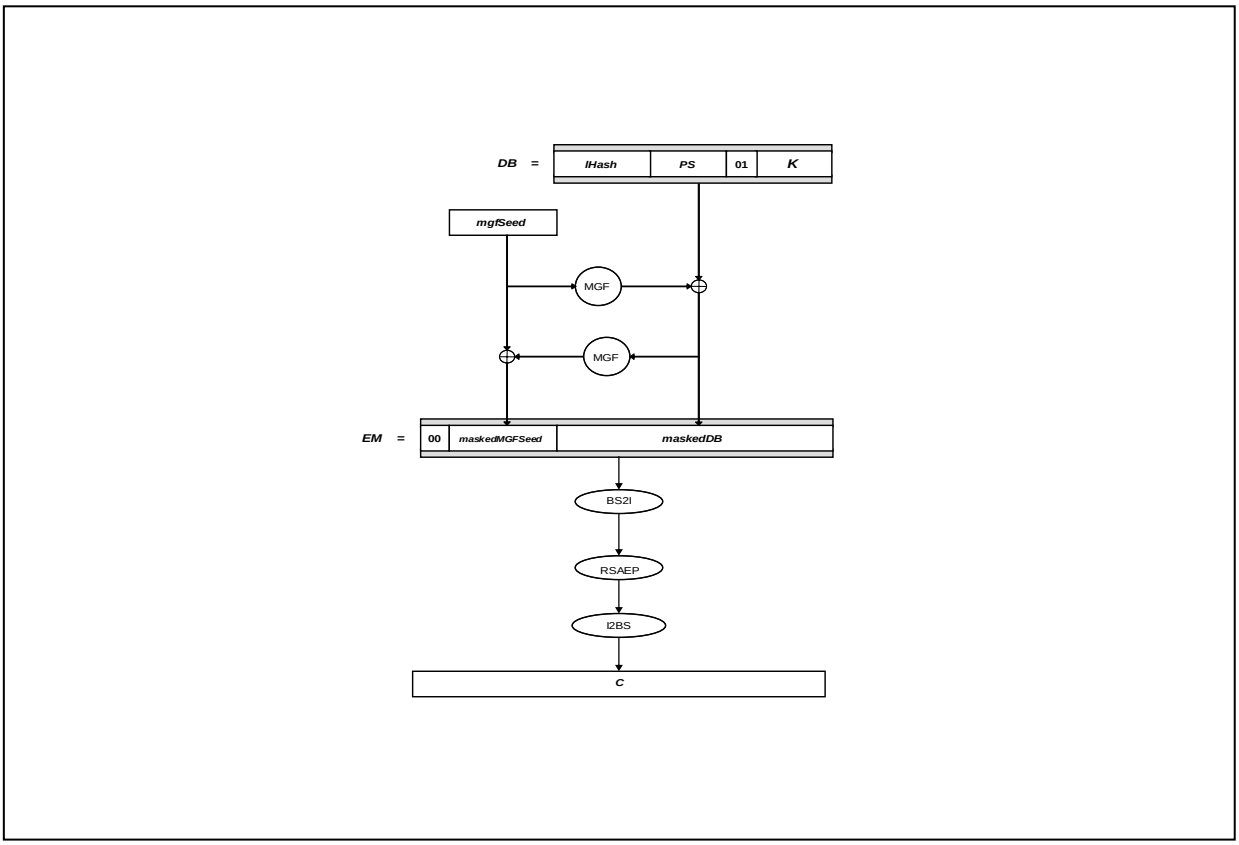

Figure 4: RSA-OAEP Encryption Operation

7.2.2.3 RSA-OAEP Decryption Operation

RSA-OAEP.DECRYPT recovers keying material from a ciphertext and additional input using an RSA private key as shown in Figure 5. Let *hlen* be the length of the hash function output in bytes.

Function call: RSA-OAEP.DECRYPT((*n*, *d*), *C*, *A*)

Input:

1. (*n*, *d*): the receiver's RSA private key.

2. *C*: the ciphertext; a byte string.

3. *A*: additional input; a byte string (may be the empty string) whose cryptographic binding to the keying material is to be verified (see Section 9.1).

Output:

K: the recovered keying material, a byte string of length at most = *nLen* – 2*hLen*-2 bytes; or,

Errors: Indications of the following:

1. Erroneous input.
2. Decryption error.

Assumptions: The RSA private key is valid.

Process:

1. Initializations:

    a. *nLen* = the length of *n* in bytes.

    b. *DecryptErrorFlag* = *False*.

2. Check for erroneous input:

    a. If *nLen* < 2*hLen* + 2, or if the length of the ciphertext *C* is not *nLen* bytes, output an indication of erroneous input and stop.

    b. Convert the ciphertext byte string *C* to a ciphertext integer *c* (see Section B.2):

    $$c = \text{BS2I}(C)$$

    c. If the ciphertext integer *c* is not such that $1 < c < n - 1$, output an indication of erroneous input and stop.

3. RSA decryption:

    a. Apply the RSADP decryption primitive (see Section 7.1.2) to the ciphertext integer *c* using the private key (*n, d*) to produce an integer *em*:

    $$em = \text{RSADP}((n, d), c).$$

    b. Convert the integer e*m* to an encoded message *EM*, a byte string of length *nLen* bytes (see Appendix B.1):

    $$EM = \text{I2BS}(em, nLen).$$

4. OAEP decoding:

a. Apply the selected hash function (see Section 5.1) to compute:

$$HA = \text{Hash}(A).$$

HA is a byte string of length *hLen* bytes.

b. Separate the encoded message *EM* into a single byte *Y*, a byte string *maskedMGFSeed′* of length *hLen* bytes, and a byte string *maskedDB′* of length *nLen – hLen –* 1 bytes as follows:

$$EM = Y \,||\, maskedMGFSeed' \,||\, maskedDB'.$$

c. Apply the mask generation function specified in Section 5.8 to compute:

$$mgfSeedMask' = \text{MGF}(maskedDB', hLen).$$

d. Let $mgfSeed' = maskedMGFSeed' \oplus mgfSeedMask'$.

e. Apply the mask generation function specified in Section 5.8 to compute:

$$dbMask' = \text{MGF}(mgfSeed', nLen - hLen - 1).$$

f. Let $DB' = maskedDB' \oplus dbMask'$.

g. Separate *DB′* into a byte string *HA′* of length *hLen* bytes and a byte string *X* of length *nLen – 2hLen –* 1 bytes as follows:

$$DB' = HA' \,||\, X.$$

5. Check for RSA-OAEP decryption errors:

   a. If *Y* is not a 00 byte, then *DecryptErrorFlag* = *True*.

   b. If *HA′* does not equal *HA*, then *DecryptErrorFlag* = *True*.

   c. If *X* does not have the form *PS* || 01 || *K*, where *PS* consists of zero or more consecutive 00 bytes, then *DecryptErrorFlag* = *True*.

   The type(s) of any error(s) found **shall not** be reported.
   (See the notes below for more information.)

6. Output of the decryption process:

   a. If *DecryptErrorFlag* = *True* then output an indication of an (unspecified) decryption error and stop. (See the notes below for more information.)

   b. Otherwise, output *K*, the portion of the byte string *X* that follows the leading 01 byte.

Notes:

1. Care must be taken to ensure that the different error conditions that may be detected in Step 5 above cannot be distinguished from one another by an opponent, whether by error message or by process timing. Otherwise, an opponent may be able to obtain useful information about the decryption of a chosen ciphertext *C*, leading to the attack observed by Manger [15]. A single error message must be employed and output the same way for each type of decryption error. There should be no difference in the observable behavior for the different RSA-OAEP decryption errors.

2. In addition, care needs to be taken to ensure that even if there are no errors, an implementation does not reveal partial information about the encoded message *EM*. For instance, the observable behavior of the mask generation function must not reveal even partial information about the MGF seed employed in the process (since that could compromise portions of the *maskedDB'* segment of *EM*). An opponent who can reliably obtain particular bits of *EM* for sufficiently many chosen ciphertexts may be able to obtain the full decryption of an arbitrary ciphertext by applying the bit-security results of Håstad and Näslund [17].

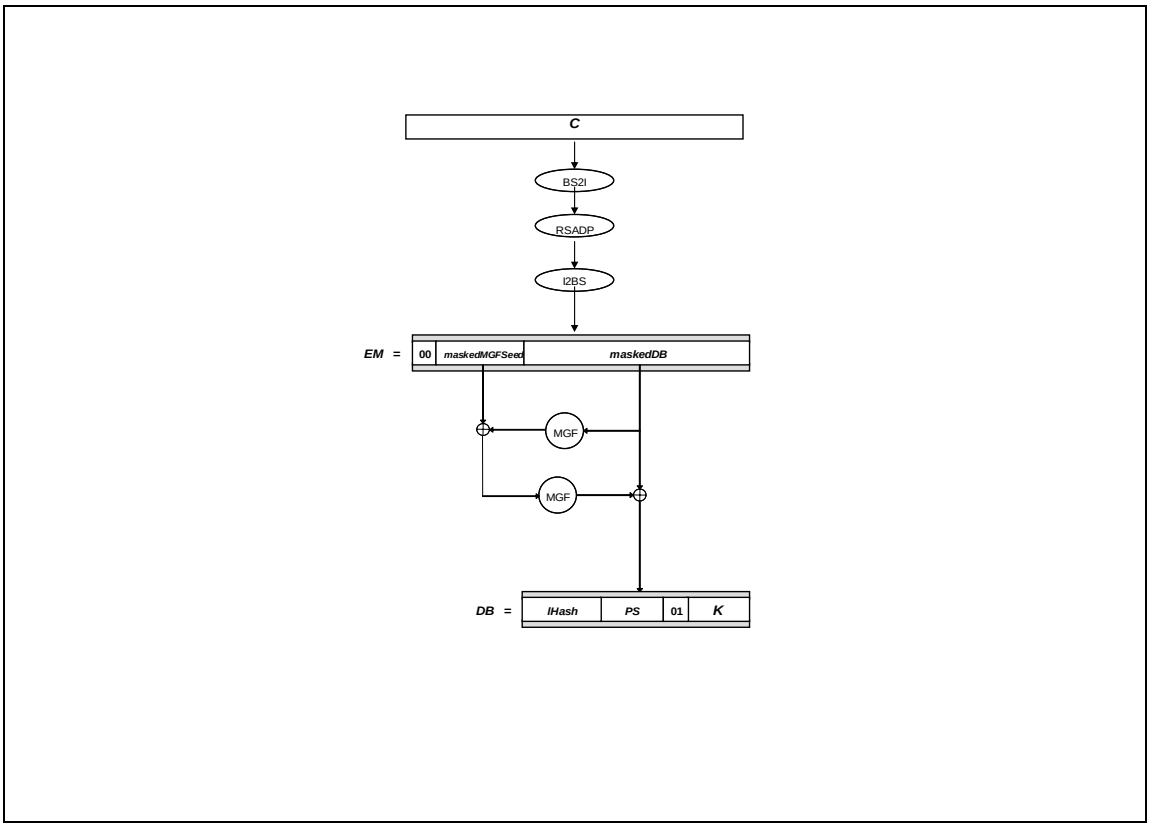

Figure 5: RSA-OAEP Decryption Operation

7.2.3 RSA-based Key-Encapsulation Mechanism with a Key-Wrapping Scheme (RSA-KEM-KWS)

RSA-KEM-KWS is used by the KTS-KEM-KWS key-transport schemes (see Section 9.3). RSA-KEM-KWS operations include a key-encapsulation method based on the RSASVE secret-value encapsulation operations and an **approved** key-derivation function (which depend, in turn, upon an **approved** random bit generator, the RSAEP and RSADP primitives, and an **approved** hash function). These operations are used to communicate a symmetric key-wrapping key to the intended receiving party. RSA-KEM-KWS operations also include an **approved** symmetric key-wrapping algorithm, which is used to convey the actual keying material to the intended receiving party.

RSA-KEM-KWS can process keying material of any length supported by the key-wrapping algorithm.

7.2.3.1 RSA-KEM-KWS Components

RSA-KEM-KWS uses the following components:

1. KDF: A key derivation function (see Section 5.9).

2. KWA: A symmetric key-wrapping algorithm, consisting of a wrapping operation KWA.WRAP and an unwrapping operation KWA.UNWRAP (see Section 5.7).

3. RSASVE: A Secret value encapsulation operation that generates and encrypts a shared secret value to produce ciphertext (using the RSASVE.GENERATE operation in Section 7.2.1.2) or recovers the shared secret value from the ciphertext (using the RSASVE.RECOVER operation in Section 7.2.1.3).

4. RBG: A random bit generator (see Section 5.3). The security strength for the RBG **shall** be at least the target security strength for the key establishment scheme.

7.2.3.2 RSA-KEM-KWS Encryption Operation

RSA-KEM-KWS.ENCRYPT is illustrated in Figure 6. The public key-establishment key of the intended receiving party is input to RSASVE.GENERATE, obtaining a secret value Z and corresponding ciphertext byte string C_0. This secret value, along with any required *OtherInfo* shared by the sender and the intended receiving party (see Section 5.9), is used as input to the KDF to obtain a key-wrapping key. This key-wrapping key, along with (optional) additional input A (see Section 9.1) that is known to both the sender and the intended receiving party, is used by the key-wrapping algorithm to encrypt the keying material, producing a ciphertext byte string C_1. The output of the RSA-KEM-KWS encryption operation is the concatenation of C_0 and C_1.

Function call: RSA-KEM-KWS.ENCRYPT((*n*, *e*), *kwkBits*, *K A*)

Input:

1. (*n*, *e*): the receiver's RSA public key.

2. *kwkBits*: the length of the key-wrapping key in bits.
3. *K*: the keying material to be wrapped, a byte string.
4. *A*: the additional input (see Section 9.1), a byte string (may be the empty string).

Output:

C: the ciphertext; a byte string.

Errors: An indication that the keying material length is not supported.

Assumptions: The RSA public key is valid and the value of *KLen* is known.

Process:

1. *nLen* = the length of *n* in bytes.
2. Length checking:
   a. *KLen* = the length of *K* in bytes.
   b. If *KLen* is not among the lengths supported by the key-wrapping algorithm, output an indication that the keying material length is not supported and stop.
3. Secret value generation and encapsulation:

   Use the RSASVE GENERATE operation specified in Section 7.2.1.2 to generate a secret value byte string *Z* and a corresponding ciphertext byte string C_0 using the responder's public key, where both *Z* and C_0 are *nLen* bytes in length.

   $$(Z, C_0) = \text{RSASVE GENERATE}((n, e)).$$

4. Key derivation:

   Derive a key-wrapping key *KWK* of length *kwkBits* bits from the byte string *Z*

   $$KWK = \text{KDF}(Z, kwkBits, OtherInfo),$$

   where the *OtherInfo* is known by both parties (see Section 5.9).

5. Key-wrapping:

   Wrap the keying material *K* (see Section 5.7) using the key-wrapping key *KWK*, associating it with the additional input, *A* to produce a KWA-ciphertext byte string C_1:

   $$C_1 = \text{KWA.WRAP}(KWK, K, A) .$$

6. Concatenation:

   Concatenate the RSA-ciphertext byte string C_0 and the KWA-ciphertext byte string C_1 to form a ciphertext byte string C:

   $$C = C_0 \| C_1.$$

7. Output the ciphertext byte string C.

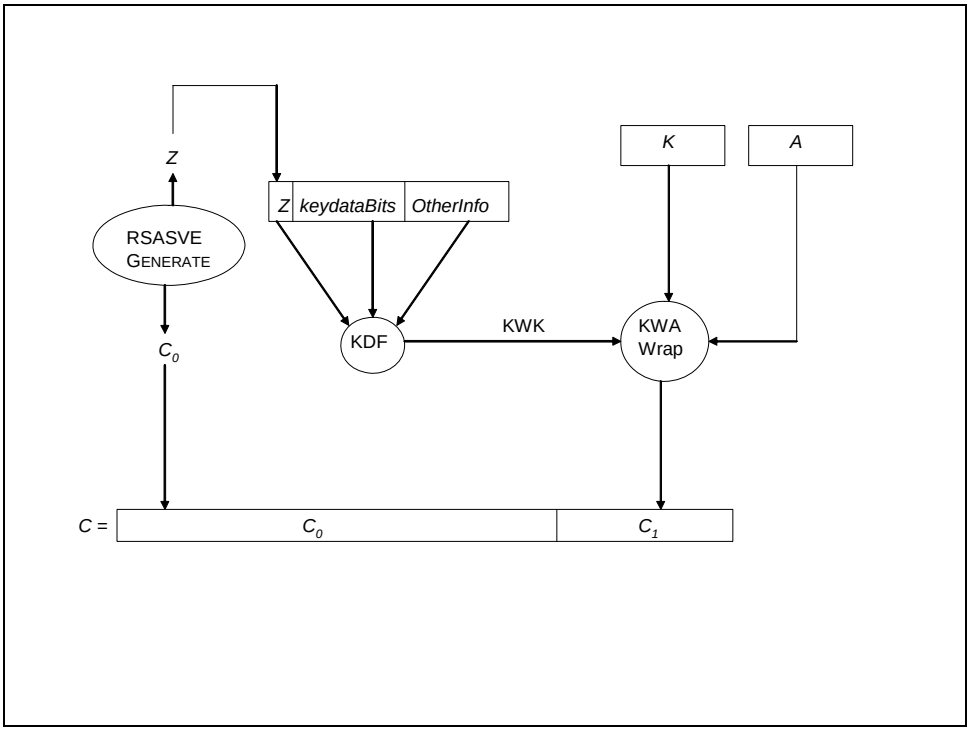

Figure 6: RSA-KEM-KWS Encryption Operation

7.2.3.3 RSA-KEM-KWS Decryption Operation

RSA-KEM-KWS.DECRYPT is illustrated in Figure 7. The private key-establishment key of the intended receiving party and C_0 are input to RSASVE.RECOVER, which returns the secret value *Z*. This secret value (along with any required *OtherInfo*) is used as input to the KDF to recover the key-wrapping key. The key-wrapping key (together with the additional data *A* – if that option was exercised) is then used to decrypt C_1 and recover the transported keying material.

Function call: RSA-KEM-KWS.DECRYPT((*n*, *d*), *C, kwkBits A*)

Input:

1. (*n*, *d*): the recipient's RSA private key..

2. *C*: the ciphertext; a byte string.
3. *kwkBits*: the length of the key-wrapping key in bits.
4. *A*: additional input; a byte string (may be the empty string).

Output:

K: the recovered keying material that was wrapped; a byte string.

Errors: An indication of a decryption error.

Assumptions: The RSA private key is valid, and the value of *KBits* is known.

Process:

1. *nLen* = the length of *n* in bytes.
2. Length checking:
   a. *cLen* = the length of the ciphertext string *C* in bytes.
   b. If *cLen* ≤ *nLen*, or if *cLen* − *nLen* is not among the lengths supported by the symmetric key-wrapping algorithm, output an indication of a decryption error and stop.
3. Separation:

   Separate the ciphertext byte string *C* into an RSA-ciphertext byte string C_0 of length *nLen* bytes and a KWA-ciphertext byte string C_1 of length *cLen* − *nLen* bytes:

   $$C = C_0 \,||\, C_1.$$

4. Recover Shared Secret:

   Recover the shared secret byte string *Z* from the ciphertext byte string C_0 using the RSASVE.RECOVER operation specified in Section 7.2.1.3.

   $$Z = \text{RSASVE.RECOVER}((n, d), C_0)$$

   If an indication of a decryption error is returned, output an indication of a decryption error and stop.

5. Key derivation:

   Derive a key-wrapping key *KWK* of length *kwkBits* bits from the byte string *Z*

   $$KWK = \text{KDF}(Z, KBits, OtherInfo),$$

where the *OtherInfo* is known by both parties (see Section 5.9).

6. Key unwrapping:

   Unwrap the KWA-ciphertext byte string C_1 using the key-wrapping key *KWK* to recover the keying material *K* (see Section 5.7), verify the correctness of *A*:

   $$K = \text{KWA.UNWRAP}(KWK, C_1\ A).$$

   If the unwrapping operation outputs an error indicator, output an indication of a decryption error and stop.

7. Output the keying material *K*.

Notes:

1. Care needs be taken to ensure that the different error conditions in Steps 2.2, 4, and 6 cannot be distinguished from one another by an opponent, whether by error message or timing. Otherwise, an opponent may be able to obtain useful information about the decryption of a chosen ciphertext *C*, leading to the attack observed by Manger [15]. A single error message **shall** be employed and output the same way for each error type. There should be no difference in timing or other behavior for the different errors. In addition, care needs to be taken to ensure that even if there are no errors, an implementation does not reveal partial information about the shared secret *Z*. An opponent who can reliably obtain particular bits of *Z* for sufficiently many chosen ciphertexts may be able to obtain the full decryption of an arbitrary ciphertext by applying the bit-security results mentioned in Annex B5.2.2 (last paragraph) of ANS X9.44.

2. In addition, care must be taken to ensure that an implementation does not reveal information about the encapsulated secret value *Z*. For instance, the observable behavior of the KDF must not reveal even partial information about the *Z* value employed in the key derivation process. An opponent who can reliably obtain particular bits of *Z* for sufficiently many chosen ciphertexts may be able to obtain the full decryption of an arbitrary ciphertext by applying the bit-security results of Håstad and Näslund [17].

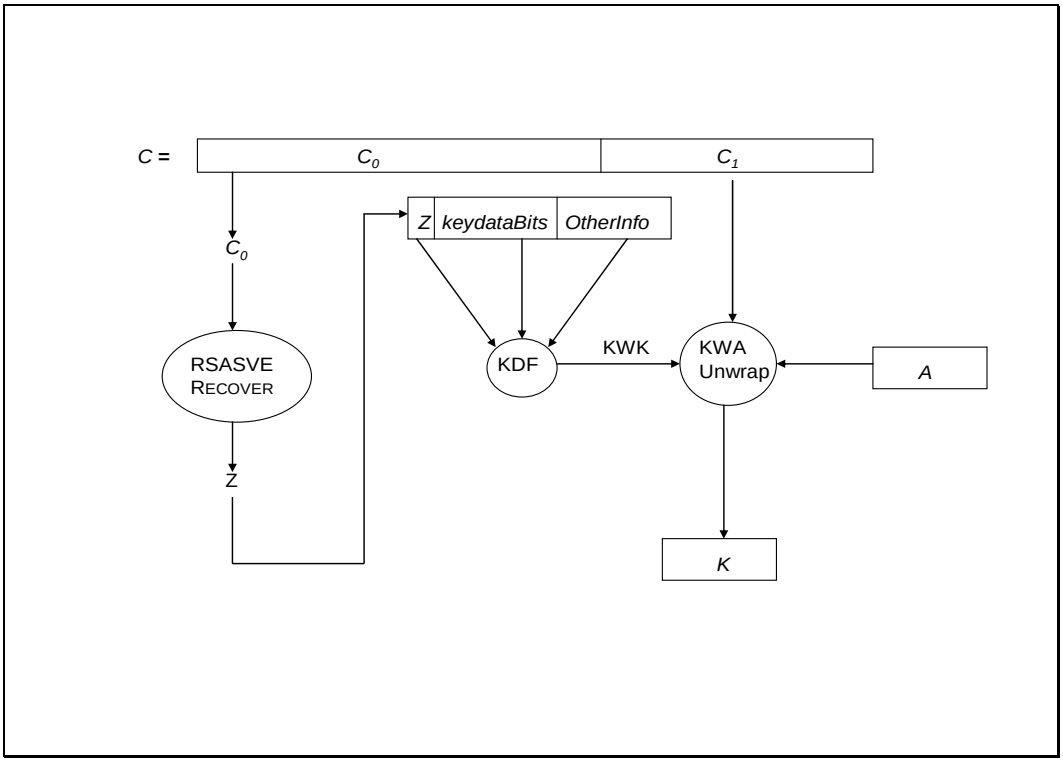

Figure 7: RSA-KEM-KWS Decryption Operation

8 Key Agreement Schemes

In a key agreement scheme, two parties, the *initiator* and the *responder*, establish keying material over which neither has direct control of the result, but both have influence. This Recommendation provides two families of key agreement schemes, KAS1 and KAS2. These schemes are based on secret value encapsulation (see Section 7.2.1).

Key confirmation is included in some of these schemes to provide assurance that the participants share the same keying material; see Section 6.6 for the details of key confirmation. When possible, each party **should** have such assurance. Although other methods are often used to provide this assurance, this Recommendation makes no statement as to the adequacy of these other methods.

The scheme initiator, party U, **shall** have an identifier ID_U, and the scheme responder, party V, **shall** have an identifier ID_V. The identifiers **shall** be non-null and selected in accordance with the protocol utilizing the scheme. When a party's public key is employed in a scheme, that party's identifier **shall** be bound to its public key.

A general flow diagram is provided for each key agreement scheme. The dotted-line arrows represent the distribution of public keys that may be distributed by the parties themselves or by a third party, such as a Certification Authority (CA). The solid-line arrows represent the distribution of nonces or cryptographically protected values that occur during the key agreement scheme. Note that the flow diagrams in this Recommendation omit explicit mention of various

validation checks that are required. The flow diagrams and descriptions in this Recommendation assume a successful completion of the key agreement process.

8.1 Common Components for Key Agreement

The key agreement schemes in this Recommendation have the following common components:

1. RSASVE: RSA secret value encapsulation, consisting of a generation operation RSASVE.GENERATE and a recovery operation RSASVE.RECOVER (see Section 7.2.1).
2. KDF: A key derivation function (see Section 5.9).

8.2 The KAS1 Family

In this family of key agreement schemes, only the responder's key-establishment key pair is used. Note that both parties may actually have key-establishment key pairs, but only the responder's key pair is used in the scheme.

The schemes in this family have the following general form:

1. Party U (the initiator) generates a secret value (which will become a shared secret) and a corresponding ciphertext using the RSASVE.GENERATE operation and party V's (the responder's) public key-establishment key, and sends the ciphertext to party V.
2. Party V recovers the shared secret from the ciphertext using the RSASVE.RECOVER operation and its private key-establishment key. Party V generates a nonce and sends it to party U.
3. Both parties then derive keying material from the shared secret and "other information", including party V's nonce, using a key derivation function. The length of the keying material that can be agreed on is limited only by the length that can be output by the key derivation function.
4. If key confirmation is incorporated, then the derived keying material is parsed into two parts, *MacKey* and *KeyData.* The *MacKey and MacData* are used to compute a *MacTag* of length *MacTagLen* (see Section 6.6.1). The *MacTag* is sent from the provider to the recipient. If the *MacTag* computed by the provider matches the *MacTag* computed by the recipient, then the successful establishment of keying material is confirmed to the recipient.

The following schemes are defined:

1. **KAS1-basic,** the basic scheme without key confirmation (see Section 8.2.2).
2. **KAS1-responder-confirmation,** a variant of **KAS1-basic** with unilateral key confirmation from party V to party U (see Section 8.2.3).

For the security properties of this family, see Section 8.2.4.

8.2.1 KAS1 Family Prerequisites

1. The responder **shall** have been designated as the owner of a key-establishment key pair that was generated as specified in Section 6.3. The responder **shall** have assurance of its possession of the correct value for its private key as specified in Section 6.5.1.
2. The initiator and responder **shall** have agreed upon an **approved** key derivation function (see Section 5.9), an **approved** hash function appropriate for use with the key derivation function and associated parameters (see Sections 5.1 and 6.2.3), the value of *KBits*, and the contents of the *OtherInfo* field used during key derivation.
3. When key confirmation is used, the initiator and responder **shall** have agreed upon an **approved** MAC algorithm and associated parameters (see Sections 5.2 and 6.2.3).
4. Prior to or during the key agreement process, each party **shall** obtain the identifier that is to be associated with the other party during the key agreement transaction. The initiator **shall** obtain the public key-establishment key that is bound to the responder's identifier in a trusted manner (e.g., from a certificate signed by a trusted CA). The initiator **shall** also obtain the assurance of validity of this public key as specified in Section 6.4.2.
5. The following is a prerequisite for using any keying material derived during a KAS1 key agreement scheme for purposes beyond those of the scheme itself.

   The initiator of a particular KAS1 key agreement transaction **shall** obtain assurance that the intended responder is (or was) in possession of the private key-establishment key corresponding to the public key-establishment key used by the initiator during that transaction, as specified in Section 6.5.2.

   This requirement recognizes the possibility that assurance of private-key possession may be provided/obtained by means of key confirmation performed as part of a particular KAS1 transaction.

8.2.2 KAS1-basic

KAS1-basic is the basic key agreement scheme in the KAS1 family. In this scheme, the responder does not contribute to the formation of the shared secret; instead, a nonce is used as a responder-selected contribution to the KDF, ensuring that both parties influence the derived keying material.

Let ($PubKey_V$, $PrivKey_V$) be party V's key-establishment key pair. Let *KBits* be the intended length in bits of the keying material to be established. The parties **shall** perform the following or an equivalent sequence of steps, as illustrated in Figure 8.

Party U **shall** execute the following key agreement steps in order to a) establish a shared secret *Z* with party V, and b) derive shared secret keying material from *Z*.

Actions: Party U **shall** derive secret keying material as follows:

1. Use the RSASVE.GENERATE operation in Section 7.2.1.2 to generated a secret value *Z* and a corresponding ciphertext *C* using party V's public key-establishment key $PubKey_V$. Note that the secret value Z will become a shared secret when recovered by Party V.

2. Send the ciphertext *C* to party V.
3. Obtain party V's nonce N_V from party V. If N_V is not available, output an error indicator and stop.
4. Construct the other information *OtherInfo* for key derivation (see Section 5.9) using (at a minimum) the identifiers ID_U and ID_V, and the nonce N_V.
5. Use the agreed-upon key derivation function (see Section 5.9) to derive secret keying material *DerivedKeyingMaterial* of length *KBits* from the shared secret *Z* and *OtherInfo*. If the key derivation function outputs an error indicator, zeroize all copies of Z, output an error indicator and stop.
6. Zeroize all copies of the shared secret *Z* and output the *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *KBits* or an error indicator.

Party V **shall** execute the following key agreement steps in order to a) establish a shared secret *Z* with party U, and b) derive shared secret keying material from *Z*.

Actions: Party V **shall** derive secret keying material as follows:

1. Receive a ciphertext *C* from party U.
2. Use the RSASVE.RECOVER operation in Section 7.2.1.3 to recover the shared secret *Z* from the ciphertext *C* using the private key-establishment key $PrivKey_V$. If the call to RSASVE.RECOVER outputs an error indicator, zeroize the result of all intermediate calculations used in the attempted recovery of *Z*, output an error indicator and stop (see Note).
3. Obtain a nonce N_V (see Section 5.6) and send N_V to party U.
4. Construct the other information *OtherInfo* for key derivation (see Section 5.9) using (at a minimum) the identifiers ID_V and ID_U, and the nonce N_V.
5. Use the agreed-upon key derivation function (see Section 5.9) to derive secret keying material *DerivedKeyingMaterial* of length *KBits* from the shared secret *Z* and *OtherInfo*. If the key derivation function outputs an error indicator, zeroize all copies of Z, output an error indicator and stop.
6. Zeroize all copies of the shared secret *Z* and output the *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *KBits* or error indicator.

Initiator (Party U)		Responder (Party V)
		($PubKey_V$, $PrivKey_V$)
Obtain responder's public key-establishment key	$PubKey_V$ ◄- - - - - - -	
(Z, C) = **RSASVE.GENERATE**($PubKey_V$)	C ——►	Z = **RSASVE.RECOVER**($PrivKey_V$, C)
Derived keying material = KDF(Z, *KBits*, *OtherInfo*)	N_V ◄——	Derived keying material = KDF(Z, *KBits*, *OtherInfo*)

Figure 8: KAS1-basic Scheme

The messages may be sent in a different order. Even though party U remains the designated initiator, N_V may be sent before C.

It is extremely important that an implementation not reveal any sensitive information. It is also important to conceal partial information about the shared secret Z.

8.2.3 KAS1 Key Confirmation

The **KAS1-responder-confirmation** scheme is based on the **KAS1-basic** scheme.

8.2.3.1 KAS1 Key Confirmation Components

The components for KAS1 key confirmation are the common components listed in Section 8.1, plus the following:

3. MAC: A message authentication code algorithm with the following parameters (see Section 5.2),
    a. *MacKeyLen*: the length in bytes of *MacKey* (see Table 1 in Section 6.2.3), and
    b. *MacTagLen*: the length in bytes of the *MacTag* (see Table 1 in Section 6.2.3).

For KAS1 key confirmation, the length of the keying material **shall** be at least *MacKeyLen* bytes, where *MacKeyLen* is the length of the *MacKey* (see Section 6.2.3), and usually longer so that other keying material is available for subsequent operations. The *MacKey* **shall** be the first *MacKeyLen* bytes of the keying material and **shall** be used only for the key confirmation operation.

8.2.3.2 KAS1-responder-confirmation

Figure 9 depicts a typical flow for a KAS1 scheme with unilateral key confirmation from party V to party U. In this scheme, party V, the scheme responder, and party U, the scheme initiator, assume the roles of key confirmation provider and recipient, respectively.

To provide (and receive) key confirmation (as described in Section 6.6.1), both parties set $EphemData_V = N_V$, and $EphemData_U = C$:

Party V provides $MacTag_V$ to party U (as specified in Section 6.6.1, with P = V and R = U), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$$MacData_V = \text{“KC\_1\_V”} \parallel ID_V \parallel ID_U \parallel N_V \parallel C\{ \parallel Text\}.$$

The recipient (party U) uses the identical format and values to compute $MacTag_V$ and then verifies that the newly computed $MacTag_V$ matches the $MacTag_V$ value provided by party V.

The *MacKey* used during Key Confirmation **shall** be zeroized by Party V immediately after the computation of $MagTag_V$, and by Party U immediately after the verification of the received $MacTag_V$ or a (final) determination that the received $MacTag_V$ is in error.

Initiator (Party U)		Responder (Party V)
		($PubKey_V$, $PrivKey_V$)
Obtain responder's public key-establishment key	$PubKey$ ←--------	
(Z, C) = RSASVE.GENERATE($PubKey_V$)	C ——→	Z = RSASVE.RECOVER($PrivKey_V$, C)
$MacKey$ || $KeyData$ = KDF(Z, $KBits$, $OtherInfo$)	N_V ←——	$MacKey$ || $KeyData$ = KDF(Z, $KBits$, $OtherInfo$)
$MacTag_V$ =? MAC($MacKey$, $MacTagLen$, $MacData_V$)	$MacTag_V$ ←——	$MacTag_V$ = MAC($MacKey$, $MacTagLen$, $MacData_V$)

Figure 9: KAS1-responder-confirmation Scheme (from Party V to Party U)

Certain messages may be combined or sent in a different order. E.g., N_V and $MacTag_V$ may be sent together. Alternatively, N_V may be sent before C even though party U remains the designated initiator.

8.2.4 KAS1 Security Properties

In each scheme included in this family, only the identifier of V (the responder) is required to be bound to a public key-establishment key. U (the initiator) has assurance that no unintended party can recover Z from C (without the compromise of private information).

The responder, however, has no such assurance. In particular, V has no assurance as to the accuracy of the identifier claimed by the initiator and, therefore, has no assurance as to the true source of the ciphertext C.

Due to the initiator's unilateral selection of the random Z value, U has assurance that fresh keying material will be derived in each instantiation of these schemes. V has similar assurance owing to its contribution of the nonce N_V to the KDF input.

A compromise of the responder's private key will allow an adversary to masquerade as the responder in future key establishment transactions, and compromise all shared secrets (and

derived keying material) resulting from past and future KAS1 transactions having that party as the responder – assuming that a malicious party has access to the publicly exchanged data (including C and N_V). Other applications that rely on the private key will be affected.

It is important to understand that a scheme may be just one of several components of a protocol. The combination of these components may endow the protocol with additional security properties beyond those provided by any particular component. Note that protocols, per se, are not specified in this Recommendation.

Through the inclusion of $MacTag_V$ in the **KAS1-responder-confirmation** scheme, and by successfully comparing the received value of $MacTag_V$ with its own computation, the initiator (U) obtains assurance that

1. the responder (V) has correctly recovered Z from C;
2. that both parties agree on the values of ID_V, ID_U, N_V, and C;
3. that (at least the *MacKey* portion of) the derived keying material has been correctly computed by V;
4. V is in possession of the correct private key that corresponds to the public key used in the transaction, and
5. V has actively participated in the transaction.

Consequently, responder authentication is implicitly provided by the binding of party V's identifier to the public key-establishment key (see Section 6.7).

8.3 KAS2 Family

In this family of key agreement schemes, both the initiator's and responder's key-establishment key pairs are used.

The schemes in this family have the following general form:

1. Party U (the initiator) generates a secret value (which will become a part of the shared secret) and a corresponding ciphertext using the RSASVE.GENERATE operation and party V's (the responder's) public key-establishment key, and sends the ciphertext to party V.
2. Party V recovers the shared secret value from the ciphertext received from party U using the RSASVE.RECOVER operation and its private key-establishment key.
3. Party V generates a secret value (which will become a second part of the shared secret) and the corresponding ciphertext using RSASVE GENERATE operation and party U's (the initiator's) public key-establishment key, and sends the ciphertext to party U.
4. Party U recovers the shared secret value from the ciphertext received from party V using the RSASVE.RECOVER operation and its private key-establishment key.

5. Both parties concatenate the shared secret values to form the shared secret, and then derive keying material from the shared secret and "other information" using a key derivation function. The length of the keying material that can be agreed on is limited only by the length that can be output by the key derivation function.

6. Party U and/or party V may additionally provide key confirmation. If key confirmation is incorporated, then the derived keying material is parsed into two parts, *MacKey* and *KeyData.* The *MacKey* is then used to compute a *MacTag* of *MacTagLen* bytes on *MacData* (see Section 6.6.1). The *MacTag* is sent from the provider to the recipient. If the *MacTag* computed by the provider matches the *MacTag* computed by the recipient, then the successful establishment of keying material is confirmed to the recipient.

The following schemes are defined:

1. **KAS2-basic,** the basic scheme without key confirmation (see Section 8.3.2).

2. **KAS2-responder-confirmation,** a variant of **KAS2-basic** with unilateral key confirmation from party V to party U (see Section 8.3.3.2).

3. **KAS2-initiator-confirmation,** a variant of **KAS2-basic** with unilateral key confirmation from party U to party V (see Section 8.3.3.3).

4. **KAS2-bilateral-confirmation,** a variant of **KAS2-basic** with bilateral key confirmation between party U and party V (see Section 8.3.3.4).

8.3.1 KAS2 Family Prerequisites

1. Each party (initiator and responder) **shall** have been designated as the owner of a key-establishment key pair that was generated as specified in Section 6.3. Prior to or during the key agreement process, each party **shall** obtain assurance of its possession of the correct value for its own private key as specified in Section 6.5.1.
2. The initiator and responder **shall** have agreed upon an **approved** key derivation function (see Section 5.9), an **approved** hash function appropriate for use with the key derivation function and associated parameters (see Sections 5.1 and 6.2.3), the value of *KBits*, and the contents of the *OtherInfo* field used during key derivation.
3. Prior to or during the key agreement process, each party **shall** obtain the identifier that is to be associated with the other party during the key agreement transaction and **shall** obtain the public key-establishment key that is bound to that identifier. These public keys **shall be** obtained in a trusted manner (e.g., from a certificate signed by a trusted CA). Each party **shall** obtain assurance of validity of the public key bound to the other party's identifier, as specified in Section 6.4.2.
4. The following is a prerequisite for using any keying material derived during a KAS2 key agreement scheme for purposes beyond those of the scheme itself.

   The recipient of a public key-establishment key that is used by the recipient during a particular KAS2 key agreement transaction **shall** obtain assurance that its (claimed) owner is (or was) in possession of the corresponding private key-establishment key, as specified in Section 6.5.2. That is,

a. The initiator of a particular KAS2 key agreement transaction **shall** obtain assurance that the intended responder is (or was) in possession of the private key-establishment key corresponding to the public key-establishment key used by the initiator during that transaction (as specified in Section 6.5.2);

b. The responder in a particular KAS2 key agreement transaction **shall** obtain assurance that the apparent initiator is (or was) in possession of the private key-establishment key corresponding to the public key-establishment key used by the responder during that transaction (as specified in Section 6.5.2).

This requirement recognizes the possibility that assurance of private-key possession may be provided/obtained by means of key confirmation performed as part of a particular KAS2 transaction.

8.3.2 KAS2-basic

Figure 10 depicts the typical flow for the **KAS2-basic** scheme. The parties exchange secret values that are concatenated together to form the mutually determined shared secret to be input to the key derivation function.

Party U **shall** execute the following key agreement steps in order to a) establish a mutually determined shared secret *Z* with party V, and b) derive secret keying material from *Z*.

Actions: Party U **shall** derive secret keying material as follows:

1. Use the RSASVE.GENERATE operation in Section 7.2.1.2 to generate a secret value Z_U and a corresponding ciphertext C_U using party V's public key-establishment key $PubKey_V$.
2. Send the ciphertext C_U to party V.
3. Receive a ciphertext C_V from party V. If C_V is not available, output an error indicator and stop.
4. Use the RSASVE.RECOVER operation in Section 7.2.1.3 to recover the Z_V from the ciphertext C_V using the private key-establishment key $PrivKey_U$. If the call to RSASVE.RECOVER outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted recovery of Z_V, zeroize Z_u, and output an error indicator and stop.
5. Construct the mutually determined shared secret *Z* from Z_U and Z_V:

$$Z = Z_U \,||\, Z_V.$$

6. Construct the other information *OtherInfo* for key derivation (see Section 5.9) using (at a minimum) the identifiers ID_U and ID_V.
7. Use the agreed-upon key derivation function (see Section 5.9) to derive secret keying material *DerivedKeyingMaterial* of length *KBits* from the shared secret *Z* and *OtherInfo*.

If the key derivation function outputs an error indicator, zeroize all copies of Z, Z_U, and Z_V, and output an error indicator and stop.

8. Zeroize all copies of Z, Z_U, and Z_V, and output the *DerivedKeyingMaterial* or an error indicator.

Output: The byte string *DerivedKeyingMaterial* of length *KBits* or error indicator.

Party V **shall** execute the following key agreement steps in order to a) establish a mutually determined shared secret Z with party U, and b) derive secret keying material from Z.

Actions: Party V **shall** derive secret keying material as follows:

1. Receive a ciphertext C_U from party U.
2. Use the RSASVE.RECOVER operation in Section 7.2.1.3 to recover Z_U from the ciphertext C_U using the private key-establishment key *PrivKey*V. If the call to RSASVE.RECOVER outputs an error indicator, zeroize the result of all intermediate calculations used in the attempted recovery of Z_U, output an error indicator and stop.
3. Use the RSASVE.GENERATE operation in Section 7.2.1.2 to generate a secret value Z_V and a corresponding ciphertext C_V using party U's public key-establishment key $PubKey_U$.
4. Send the ciphertext C_V to party U.
5. Determine the mutually determined shared secret Z from Z_U and Z_V:

   $Z = Z_U \,||\, Z_V$.

6. Construct the other information *OtherInfo* for key derivation (see Section 5.9) using (at a minimum) the identifiers ID_V and ID_U.
7. Use the agreed-upon key derivation function (see Section 5.9) to derive secret keying material *DerivedKeyingMaterial* of length *KBits* from the shared secret Z and *OtherInfo*. If the key derivation function outputs an error indicator, zeroize all copies of Z, Z_U, and Z_V, and output an error indicator and stop.
8. Zeroize all copies of Z, Z_U, and Z_V, and output the *DerivedKeyingMaterial*.

Output: The byte string *DerivedKeyingMaterial* of length *KBits* or error indicator.

Initiator (Party U)		**Responder (Party V)**
($PubKey_U$, $PrivKey_U$)		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ← — — —	
	$PubKey_U$ — — — →	Obtain party U's public key-establishment key
(Z_U, C_U) = RSASVE.GENERATE($PubKey_V$)	C_U ——→	Z_U = RSASVE.RECOVER($PrivKey_V$, C_U)
Z_V = RSASVE.RECOVER($PrivKey_U$, C_V)	C_V ←——	(Z_V, C_V) = RSASVE.GENERATE($PubKey_U$)
$Z = Z_U \| Z_V$		$Z = Z_U \| Z_V$
DerivedKeyingMaterial = KDF(Z, *KBits*, *OtherInfo*)		*DerivedKeyingMaterial* = KDF(Z, *KBits*, *OtherInfo*)

Figure 10: KAS2-basic Scheme

The messages may be sent in a different order. Even though party U remains the designated initiator, C_V may be sent before C_U.

It is extremely important that an implementation not reveal any sensitive information. It is also important to conceal partial information about Z_U, Z_V and Z to prevent chosen-ciphertext attacks on the secret value encapsulation scheme. In particular, the observable behavior of the key-agreement process **should not** reveal partial information about the shared secret Z.

8.3.3 KAS2 Key Confirmation

The KAS2 key confirmation schemes are based on the **KAS2-basic** scheme.

8.3.3.1 KAS2 Key Confirmation Components

The scheme components for KAS2 key confirmation are the common components in Section 8.1, plus the following:

3. MAC: A message authentication code algorithm (see Section 5.2)
    a. *MacKeyLen*: the length in bytes of the *MacKey* (see Table 1 in Section 6.2.3).

b. *MacTagLen*: the length in bytes of the *MacTag* (see Table 1 in Section 6.2.3).

For this scheme, the length of the keying material **shall** be at least *MacKeyLen,* where *MacKeyLen* is the length in bytes of the *MacKey* (see Section 6.2.3), and usually longer so that other keying material is available for subsequent operations. The *MacKey* **shall** be the first *MacKeyLen* bytes of the keying material and **shall** be used only for the key confirmation operation.

8.3.3.2 KAS2-responder-confirmation

Figure 11 depicts a typical flow for a KAS2 scheme with unilateral key confirmation from party V to party U. In this scheme, party V, the scheme responder, and party U, the scheme initiator, assume the roles of the key confirmation provider and recipient, respectively.

To perform key confirmation (as described in Section 6.6.1), both parties set $EphemData_V = C_V$, and $EphemData_U = C_U$.

Party V provides $MacTag_V$ to party U (as specified in Section 6.6.1, with P = V and R = U), where $MacTag_V$ is computed (as specified in Section 5.2.1) on

$$MacData_V = \text{“KC\_1\_V”} \| ID_V \| ID_U \| C_V \| C_U \{ \| Text \}.$$

The recipient (party U) uses the identical format and values to compute $MacTag_V$ and then verifies that the newly computed $MacTag_V$ equals the $MacTag_V$ value provided by party V.

The *MacKey* used during key confirmation **shall** be zeroized by Party V immediately after the computation of $MagTag_V$, and by Party U immediately after the verification of the received $MacTag_V$ or a (final) determination that the received $MacTag_V$ is in error.

Initiator (Party U)		Responder (Party V)
($PubKey_U$, $PrivKey_U$)		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ← — — —	
	$PubKey_U$ — — — →	Obtain party U's public key establishment-key
(Z_U, C_U) = RSASVE.Generate($PubKey_V$)	C_U ——————→	Z_U = RSASVE.Recover($PrivKey_V$, C_U)
Z_V = RSASVE.Recover($PrivKey_U$, C_V)	C_V ←——————	(Z_V, C_V) = RSASVE.Generate($PubKey_U$)

$Z = Z_U \| Z_V$		$Z = Z_U \| Z_V$
K= KDF(Z, *KBits*, *OtherInfo*) = *MacKey* || *KeyData*		K = KDF(Z, *KBits*, *OtherInfo*) = *MacKey* || *KeyData*
$MacTag_V$ =? MAC(*MacKey*, *MacTagLen*, $MacData_V$)	$MacTag_V$ ←	$MacTag_V$ = MAC(*MacKey*, *MacTagLen*, $MacData_V$)

Figure 11: KAS2-responder-confirmation Scheme (from Party V to Party U)

Certain messages may be combined or sent in a different order. E.g., C_V and $MacTag_V$ may be sent together. Alternatively, C_V may be sent before C_U, even though party U remains the designated initiator.

8.3.3.3 KAS2-initiator-confirmation

Figure 12 depicts a typical flow for a KAS2 scheme with unilateral key confirmation from party U to party V. In this scheme, party U, the scheme initiator, and party V, the scheme responder, assume the roles of key confirmation provider and recipient, respectively.

To provide (and receive) key confirmation (as described in Section 6.6.1), both parties set $EphemData_V = C_V$, and $EphemData_U = C_U$.

Party U provides $MacTag_U$ to party V (as specified in Section 6.6.1, with P = U and R = V), where $MacTag_U$ is computed (as specified in Section 5.2.1) on

$$MacData_U = \text{“KC\_1\_U”} \| ID_U \| ID_V \| C_U \| C_V \{ \| Text \}.$$

The recipient (party V) uses the identical format and values to compute $MacTag_U$ and then verifies that the newly computed $MacTag_U$ matches the $MacTag_U$ value provided by party U.

The *MacKey* used during key confirmation **shall** be zeroized by Party U immediately after the computation of $MagTag_U$, and by Party V immediately after the verification of the received $MacTag_U$ or a (final) determination that the received $MacTag_U$ is in error.

Initiator (Party U)		**Responder (Party V)**
($PubKey_U$, $PrivKey_U$)		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ←--------	
	$PubKey_U$ -------→	*Obtain party U's public key-establishment key*
(Z_U, C_U) = RSASVE.GENERATE($PubKey_V$)	C_U ——→	Z_U = RSASVE.RECOVER($PrivKey_V$, C_U)
Z_V = RSASVE.RECOVER($PrivKey_U$, C_V)	C_V ←——	(Z_V, C_V) = RSASVE.GENERATE($PubKey_U$)
$Z = Z_U \,||\, Z_V$		$Z = Z_U \,||\, Z_V$
MacKey || *KeyData* = KDF(*Z*, *KBits*, *OtherInfo*)		*MacKey* || *KeyData* = KDF(*Z*, *KBits*, *OtherInfo*)
$MacTag_U$ = MAC(*MacKey*, *MacTagLen*, $MacData_U$)	$MacTag_U$ ——→	$MacTag_U$ =? MAC(*MacKey*, *MacTagLen*, $MacData_U$)

Figure 12: KAS2-initiator-confirmation Scheme (from Party U to Party V)

Certain messages may be sent in a different order (and combined with others). Even though party U remains the designated initiator, C_V may be sent before C_U; in which case C_U and $MacTag_U$ may be sent together.

8.3.3.4 KAS2-bilateral-confirmation

Figure 13 depicts a typical flow for a KAS2 scheme with bilateral key confirmation. In this scheme, party U, the scheme initiator, and party V, the scheme responder, assume the roles of both the provider and the recipient in order to obtain bilateral key confirmation.

To provide bilateral key confirmation (as described in Section 6.6.2), party U and party V exchange and verify *MacTags* that have been computed (as specified in Section 6.6.1) using $EphemData_U = C_U$, and $EphemData_V = C_V$.

Party V provides $MacTag_V$ to party U (as specified in Section 6.6.1, with P = V and R = U); $MacTag_V$ is computed by party V (and verified by party U) on

$$MacData_V = \text{“KC\_2\_V”} \| ID_V \| ID_U \| C_V \| C_U \{ \| Text_1 \}.$$

Party U provides $MacTag_U$ to party V (as specified in Section 6.6.1, with P = U and R = V); $MacTag_U$ is computed by party U (and verified by party V) on

$$MacData_U = \text{“KC\_2\_U”} \| ID_U \| ID_V \| C_U \| C_V \{ \| Text_2 \}.$$

The *MacKey* used during key confirmation **shall** be zeroized by each party immediately following its use to compute and verify the *MacTag* for key confirmation. Once Party U has computed $MacTag_U$ and has either verified the received $MacTag_V$ or made a (final) determination that the received $MacTag_U$ is in error, Party U **shall** immediately zeroize its copy of *MacKey*. Similarly, after Party V has computed $MacTag_V$ and has either verified the received $MacTag_U$ or made a (final) determination that the received $MacTag_U$ is in error; Party V **shall** immediately zeroize its copy of *MacKey*.

Initiator (Party U)		**Responder (Party V)**
($PubKey_U$, $PrivKey_U$)		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ←– – – – – – – – – –	(Z, C) = RSASVE.GENERATE($PubKey_V$)
	$PubKey_U$ – – – – – →	Obtain party U's public key-establishment key
(Z_U, C_U) = RSASVE.GENERATE($PubKey_V$)	C_U ——————→	Z_U = RSASVE.RECOVER($PrivKey_V$, C_U)
Z_V = RSASVE.RECOVER($PrivKey_U$, C_V)	C_V ←——————	(Z_V, C_V) = RSASVE.GENERATE($PubKey_V$)
$Z = Z_U \| \| Z_V$		$Z = Z_U \| \| Z_V$
MacKey || *KeyData* = KDF(Z, *KBits*, *OtherInfo*)		*MacKey* || *KeyData*= KDF(Z, *KBits*, *OtherInfo*)
$MacTag_V$ =? MAC(*MacKey*, *MacTagLen*, $MacData_V$)	$MacTag_V$ ←——————	$MacTag_V$ = MAC(*MacKey*, *MacTagLen*, $MacData_V$)

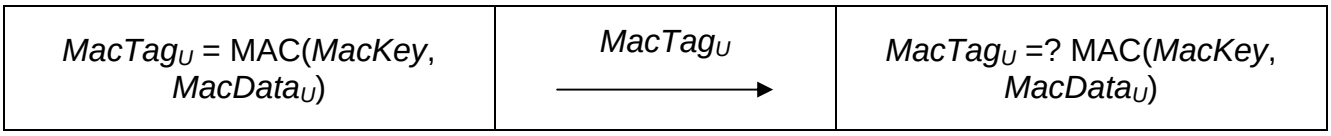

Figure 13: KAS2-bilateral-confirmation Scheme

Certain messages may be sent in a different order (and/or combined with others). Even though party U remains the designated initiator, C_V may be sent before C_U and/or $MacTag_V$ may be sent before $MacTag_U$. If C_U is sent immediately before $MacTag_U$, then C_U and $MacTag_U$ may be sent together. If C_V is sent immediately before $MacTag_V$, then C_V and $MacTag_V$ may be sent together.

8.3.4 KAS2 Security Properties

In the schemes included in this family, each party has an identifier that is bound to a public key-establishment key. Therefore, U (the initiator) has assurance that no unintended party can recover Z_U from C_U, and V (the responder) has assurance that no unintended party can recover Z_V from C_V (without the compromise of private information). Consequently, U and V both have assurance that they are the only two parties capable of deriving the keying material corresponding to $Z_U \| Z_V$.

By virtue of their random contributions (Z_U by U and Z_V by V) to the KDF input, each party also has assurance that fresh keying material will be derived in each instantiation of these schemes.

The compromise of one party's private key will allow an adversary to masquerade as that party in future key establishment transactions. However, the compromise of the private key of a single participant will not, by itself, permit the compromise of keying material derived in KAS2 transactions (between honest parties).

Through the inclusion of $MacTag_V$ in the **KAS2-responder-confirmation** and **KAS2-bilateral-confirmation** schemes, and by successfully comparing the received value of $MacTag_V$ with its own computation, the initiator (U) obtains assurance that

1. the responder (V) has correctly recovered Z_U from C_U;
2. that both parties agree on the values of ID_V, ID_U, C_V, and C_U;
3. that (at least the *MacKey* portion of) the derived keying material has been correctly computed by V;
4. V is in possession of the correct private key that corresponds to the public key used by Party U in the transaction;
5. V has actively participated in the process; and
6. U has correctly recovered Z_V from C_V and therefore possesses the correct value for its private key.

Through the inclusion of $MacTag_U$ in the **KAS2-initiator-confirmation** and **KAS2-bilateral-confirmation** schemes, and by successfully comparing the received value of $MacTag_V$ with its own computation, the responder (V) obtains assurance that

1. the initiator (U) has correctly recovered Z_V from C_V;

2. that both parties agree on the values of ID_V, ID_U, C_V, and C_U;
3. that (at least the *MacKey* portion of) the derived keying material has been correctly computed by U;
4. U is in possession of the correct private key that corresponds to the public key used by Party V in the transaction;
5. U has actively participated in the process; and
6. V has correctly recovered Z_U from C_U and, therefore, possesses the correct value for its private key.

9 IFC based Key Transport Schemes

In a key transport scheme, two parties, the *sender* and the *receiver*, establish keying material selected initially by the sender. The keying material may be cryptographically bound to additional input (see Section 9.1).

Two families of key transport schemes are specified: KTS-OAEP and KTS-KEM-KWS.

Key confirmation is included in some of these schemes to provide assurance to the sender that the participants share the same keying material (see Section 6.6 for further details on key confirmation.).

The keying material to be transported is determined by the sender in a key transport scheme and has the general form:

TransportedKeyingMaterial = *MacKey* || *KeyData*.

In key transport schemes that provide key confirmation (see Sections 9.2.4.2 and 9.3.4.2), the transported keying material **shall** contain a *MacKey* as the first bits of the keying material; the *MacKey* will be used for the computation and verification of the *MacTag*. *KeyData* is the keying material that follows the *MacKey*. The *MacKey* **shall** be generated anew for each instance of a key establishment transaction using an **approved** random bit generator at the security strength required for the key establishment transaction. The *MacKey* length **shall** be equal to or greater than the security strength associated with the modulus used in the key establishment scheme (see SP 800-57-Part 1 [7]). The *KeyData* may be null, or may contain keying material to be used subsequent to the key transport transaction. The *MacKey* **shall** be used during Key Confirmation and then immediately zeroized.

In key transport schemes that do not provide key confirmation (see Sections 9.2.4.1 and 9.3.4.1), the *TransportedKeyingMaterial* = *KeyData*. The *KeyData* contains keying material to be used subsequent to the key transport transaction.

A general flow diagram is provided for each key transport scheme. The dotted-line arrows represent the distribution of public keys that may be distributed by the parties themselves or by a third party, such as a Certification Authority (CA). The solid-line arrows represent the distribution of cryptographically protected values that occur during the key transport or key confirmation process. Note that the flow diagrams in this Recommendation omit explicit mention

of various validation checks that are required. The flow diagrams and descriptions in this Recommendation assume a successful completion of the key transport process.

9.1 Additional Input

Additional input, *A*, is supported by the key transport schemes specified in Section 9.2 and 9.3.

1. It may include a representation of shared information either exchanged by the parties or obtained from higher-level protocols, such as:

    a. the names or other identifying information (e.g., e-mail address, etc.) of the sender and receiver;

    b. nonces or other fresh data contributed by the parties;

    c. the type, length, or intended use of the keying material (and/or of individual keys within the keying material, if the keying material consists of more than one key);

    d. a counter value;

    e. secret data shared by the parties;

    f. a hash value, and/or

    g. other public data shared by the parties.

2. *A* may consist of an empty string.

3. Each party to the key establishment **shall** know the contents of *A* before it is required by the scheme.

One purpose of the additional input could be to enable the sender to indicate that it intends to employ the keying material in a specified context and to bind the keying material to this context.

The method for formatting and distributing the additional input is application-defined.

9.2 KTS-OAEP Family: Key Transport Using RSA-OAEP

The KTS-OAEP family of key transport schemes is based on RSA-OAEP encrypt and decrypt operations (see Section 7.2.2), which are, in turn, based on the asymmetric encryption and decryption primitives, RSAEP and RSADP (see Section 7.1). In this family, only the receiver's key pair is used.

The key transport schemes of this family have the following general form:

1. Party U (the sender) encrypts the keying material to be transported using the RSA-OAEP ENCRYPT operation and party V's (the receiver's) public key-establishment key to produce a ciphertext, and sends the ciphertext to party V.

2. Party V decrypts the ciphertext using its private key-establishment key and the RSA-OAEP DECRYPT operation to recover the transported keying material.

3. If key confirmation is incorporated, then the transported keying material is parsed into two parts, a transaction-specific (random) *MacKey* followed by *KeyData.* The *MacKey* portion of the keying material and an **approved** MAC algorithm are used by each party to compute a *MacTag* (of an appropriate, agreed-upon length) on what should be the same *MacData* (see Section 6.6.1). The *MacTag* computed by the key-confirmation provider (V) is sent to the key-confirmation recipient (U). If the value of the *MacTag* sent by V matches the *MacTag* value computed by U, then U obtains a confirmation of the success of the key-transport transaction.

The common components of the schemes in the KTS-OAEP family are listed in Section 9.2.2. The following schemes are then defined:

1. **KTS-OAEP-basic**, the basic scheme without key confirmation (see Section 9.2.3).

2. **KTS-OAEP -receiver-confirmation**, a variant of **KTS-OAEP-basic** with unilateral key confirmation from party V to party U (see Section 9.2.4).

For the security attributes of the KTS-OAEP family, see Section 9.2.5.

9.2.1 KTS-OAEP Family Prerequisites

1. The receiver **shall** have been designated as the owner of a key-establishment key pair that was generated as specified in Section 6.3. The receiver **shall** have assurance of its possession of the correct value for its private key as specified in Section 6.5.1.
2. The sender and receiver **shall** have agreed upon an **approved** hash function appropriate for use with the mask generation function used by RSA-OAEP (see Sections 5.1, 5.8, 6.2.3, and 7.2.2).
3. Prior to or during the transport process, the sender and receiver **shall** have either agreed upon the form and content of the additional input *A* (a byte string to be cryptographically bound to the transported keying material in that the cipher is a cryptographic function of both values), or agreed that *A* will be an empty string (see Section 9.1 above).
4. When key confirmation is used, the sender and receiver **shall** have agreed upon an **approved** MAC algorithm and associated parameters (see Sections 5.2 and 6.2.3).
5. Prior to or during the key transport process, each party **shall** obtain the identifier that is to be associated with the other party during the key transport transaction. The sender **shall** obtain the public key-establishment key that is bound to the receiver's identifier. The sender **shall** obtain this public key in a trusted manner (e.g., from a certificate signed by a trusted CA). The sender **shall** obtain assurance of validity of this public key as specified in Section 6.4.2.
6. Prior to or during the key transport process, the sender **shall** obtain assurance that the intended receiver is (or was) in possession of the (correct) private key corresponding to

the public key-establishment key used during the transaction, as specified in Section 6.5.2.

7. Prior to or during the key transport process, the keying material to be transported **shall** be determined as specified at the beginning of Section 9.

9.2.2 Common components

The schemes in the KTS-OAEP family have the following common component:

1. RSA-OAEP: asymmetric operations, consisting of an encryption operation RSA-OAEP.ENCRYPT and a decryption operation RSA-OAEP.DECRYPT (see Section 7.2.2).

9.2.3 KTS-OAEP-basic

KTS-OAEP-basic is the basic key transport scheme in the KTS-OAEP family without key confirmation.

Let ($PubKey_V$, $PrivKey_V$) be party V's (the receiver's) key-establishment key pair. Let K be the keying material to be transported from party U (the sender) to party V. The parties **shall** perform the following or an equivalent sequence of steps, which are also illustrated in Figure14.

Party U **shall** execute the following steps in order to transport keying material to party V.

Party U Actions:

1. Encrypt the keying material K using party V's public key-establishment key $PubKey_V$ and the additional input A, to produce a ciphertext C (see Section 7.2.2.2):

   C = RSA-OAEP.ENCRYPT($PubKey_V$, K, A).

2. Send the ciphertext C to party V.

Party V **shall** execute the following steps when receiving keys transported from party V.

Party V Actions:

1. Receive the ciphertext C.

2. Decrypt the ciphertext C using the private key-establishment key $PrivKey_V$ and the additional input A, to recover the transported keying material K (see Section 7.2.2.3):

   K = RSA-OAEP.DECRYPT($PrivKey_V$, C, A).

   If the decryption operation outputs an error indicator, output an error indication and stop.

Output: The byte string K or an error indicator.

Sender (Party U)		**Receiver (Party V)**
K to be transported		(*PubKey*$_V$, *PrivKey*$_V$)
Obtain party V's public key-establishment key	*PubKey*$_V$ ← — — —	
C = RSA-OAEP. ENCRYPT(*PubKey*$_V$, *K*, *A*)	*C* ——————→	*K* = RSA-OAEP. DECRYPT(*PrivKey*$_V$, *C*, *A*)

Figure 14: KTS-OAEP-basic Scheme

9.2.4 KTS-OAEP Key Confirmation

The **KES-OAEP-receiver-confirmation** scheme is based on the **KTS-OAEP-basic scheme**.

9.2.4.1 KTS-OAEP Common Components for Key Confirmation

The components for **KTS-OAEP key confirmation** are the same as for **KTS-OAEP-basic** (see Section 9.2.2), plus the following:

2. MAC: A message authentication code algorithm (see Section 5.2).

    a. *MacKeyLen*: the length in bytes of the *MacKey* (see Table 1 in Section 6.2.3).

    b. *MacTagLen*: the length in bytes of the *MacTag* (see Table 1 in Section 6.2.3).

For this scheme, the length of the keying material **shall** be at least *MacKeyLen,* where *MacKeyLen* is the length in bytes of the *MacKey* (see Section 6.2.3) and usually longer so that other keying material is available for subsequent operations. The *MacKey* **shall** be the first *MacKeyLen* bytes of the keying material and **shall** be used only for the key confirmation operation.

9.2.4.2 KTS-OAEP-receiver-confirmation

KTS-OAEP-receiver-confirmation is a variant of **KTS-OAEP-basic** with unilateral key confirmation from party V to party U.

Figure 15 depicts a typical flow for the **KTS-OAEP-receiver-confirmation** scheme. In this scheme, party V, the receiver, and party U, the sender, assume the roles of key confirmation provider and recipient, respectively.

To provide (and receive) key confirmation (as described in Section 6.6.1), both parties form *MacData* with $EphemData_V = Null$, and $EphemData_U = C$:

Party V provides $MacTag_V$ to party U (as specified in Section 6.6.1, with P = V and R = U), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$$MacData_V = \text{“KC\_1\_V”} \| ID_V \| ID_U \| Null \| C\{ \| Text\}.$$

Party U uses the identical format and values to compute $MacTag_V$ and then verifies that the newly computed $MacTag_v$ matches the $MacTag_V$ value provided by party V.

The *MacKey* used during key confirmation **shall** be zeroized by Party V immediately after the computation of $MagTag_V$, and by Party U immediately after the verification of the received $MacTag_V$ or a (final) determination that the received $MacTag_V$ is in error.

Sender (Party U)		**Receiver (Party V)**
K = *MacKey* ll *KeyData*		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ← — — —	
C = RSA-OAEP. ENCRYPT($PubKey_V$, K, A)	C ——————→	K = RSA-OAEP. DECRYPT($PrivKey_V$, C, A)
		MacKey || *KeyData* = K
$MacTag_V$ =? MAC(*MacKey*, *MacTagLen*, $MacData_V$)	$MacTag_V$ ←——————	$MacTag_V$ = MAC(*MacKey*, *MacTagLen*, $MacData_V$)

Figure 15: KET-OAEP-receiver-confirmation Scheme

9.2.5 KTS-OAEP Security Properties

In each scheme included in this family, only the identifier of V (the receiver) is bound to a public key-establishment key. U (the sender) has assurance that no unintended party can recover K from the ciphertext C (without the compromise of private information).

The receiver, however, has no such assurance. In particular, V has no assurance as to the accuracy of the identifier claimed by the sender and, therefore, has no assurance as to the true source of the ciphertext C (or the transported K).

Due to the sender's unilateral selection of K, U has assurance that fresh keying material has been transported. V has no such assurance.

A compromise of the receiver's private key will allow an adversary to masquerade as the receiver in future key establishment transactions, and compromises all keying material transported to the receiver in both past and future transactions.

In the key confirmation case, if one assumes that the transported keying material includes a *MacKey* that is (as required by this Recommendation) unique to the particular KTS-OAEP-receiver-confirmation transaction between U and V, then by successfully comparing the received value of $MacTag_V$ with its own computation, the sender (U) can obtain these assurances:

1. the receiver (V) has correctly recovered (at least the *MacKey* portion of) *K* from *C*;
2. both parties agree on the values of ID_V, ID_U, and *C*;
3. V possesses the correct value of the private key corresponding to the public key used in the transaction; and
4. V has actively participated in the transaction.

If U has transported the same *K* to multiple parties, and/or (in violation of this Recommendation) U has re-used a *MacKey*, then the return of a correct $MacTag_V$ value to U does not provide assurance that V has correctly obtained the keying material (or anything else). Anyone in possession of (at least the *MacKey* portion of) *K* could have computed $MacTag_V$.

9.3 KTS-KEM-KWS Family: Key Transport using RSA-KEM-KWS

The KTS-KEM-KWS family of key transport schemes is based on the RSA-KEM-KWS encrypt and decrypt operations. These operations employ the asymmetric RSASVE secret-value encapsulation operations and an **approved** KDF to establish a key-wrapping key that is transaction-specific. The key-wrapping key is used with an **approved** symmetric key-wrapping algorithm to wrap (and unwrap) the keying material to be transported. In this family, only party V's key pair is used.

The key transport schemes of this family have the following general form:

1. Using the RSA-KEM-KWS.ENCRYPT operation, party U (the sender) first generates a secret byte string *Z* and a corresponding ciphertext component by employing the RSASVE.GENERATE operation and the public key-establishment key of party V (the receiver). The byte string Z (along with *OtherInfo* available to U and V) is then used as input to the KDF, to derive a transaction-specific key-wrapping key (KWK) of an appropriate, agreed-upon bit length *kwkBits*. The keying material to be transported is wrapped using the KWK and the symmetric key-wrapping algorithm to produce a second ciphertext component. The two ciphertext components are sent to party V.

2. Using the RSA-KEM-KWS Decrypt operation, Party V begins by employing the RSASVE.RECOVER operation and its private key-establishment key to obtain *Z* from the first ciphertext component. Party V then employs the KDF (with inputs Z, *kwkBits*, and *OtherInfo*) to derive the same KWK that was used by U. The KWK and the symmetric key-unwrapping algorithm are used to obtain the transported keying material from the second ciphertext component.

3. If key confirmation is incorporated, the transported keying material consists of a transaction-specific (random) *MacKey* followed by *KeyData*. The *MacKey* portion of the keying material and an **approved** MAC algorithm are used by each party to compute a

MacTag (of an appropriate, agreed-upon length) on what should be the same *MacData* (see Section 6.6.1). The *MacTag* computed by the key-confirmation provider (V) is sent to the key-confirmation recipient (U). If the value of the *MacTag* sent by V matches the *MacTag* value computed by U, then U obtains a confirmation of the success of the key-transport transaction.

Common components of the schemes in the KTS-KEM-KWS family are listed in Section 9.3.2. Two schemes are then defined:

1. **KTS-KEM-KWS-basic**, the basic scheme without key confirmation (see Section 9.3.3).

2. **KTS-KEM-KWS-receiver-confirmation**, a variant with unilateral key confirmation from the receiver (Party V) to the sender (Party U) (see Section 9.3.4).

For the security attributes of the KTS-KEM-KWS family, see Section 9.3.5.

9.3.1 KTS-KEM-KWS Family Prerequisites

1. The receiver **shall** have been designated as the owner of a key-establishment key pair that was generated as specified in Section 6.3. The receiver **shall** obtain assurance of the validity of its key pair as specified in Section 6.4.1, and **shall** obtain assurance of its possession of the correct value for its private key as specified in Section 6.5.1.
2. The sender and receiver **shall** have agreed upon an **approved** key derivation function, an **approved** hash function appropriate for use with the key derivation function, and associated parameters (see Sections 5.1, 5.9, and 6.2.3).
3. The sender and receiver **shall** have agreed upon an **approved** symmetric key-wrapping algorithm and key length (*kwkBits*) employing an **approved** block cipher algorithm whose security strength is equal to or greater than the target security strength of the applicable key transport scheme (see Sections 5.7 and 7.2.3).
4. Prior to or during the transport process, the sender and receiver **shall** have either agreed upon the form and content of the additional input *A* (a byte string to be cryptographically bound to the transported keying material in that the cipher is a cryptographic function of both values), or agreed that *A* will be an empty string (see Section 9.1 above).
5. When key confirmation is used, the sender and receiver **shall** have agreed upon an **approved** MAC algorithm and associated parameters (see Sections 5.2 and 6.2.3).
6. Prior to or during the key transport process, each party **shall** obtain the identifier that is to be associated with the other party during the key transport transaction. The sender **shall** obtain the public key-establishment key that is bound to the receiver's identifier in a trusted manner (e.g., from a certificate signed by a trusted CA). The sender **shall** obtain assurance of the validity of this public key as specified in Section 6.4.2.
7. Prior to or during the key transport process, the sender **shall** obtain assurance that the intended receiver is (or was) in possession of the private key corresponding to the public key-establishment key used during the transaction, as specified in Section 6.5.2.
8. Prior to or during the key transport process, the keying material to be transported **shall** be determined as specified in Section 9.

9.3.2 Common Components of the KTS-KEM-KWS Schemes

The schemes in the KTS-KEM-KWS family have the following common component:

1. RSA-KEM-KWS: Consisting of an encryption operation RSA-KEM-KWS.ENCRYPT and a decryption operation RSA-KEM-KWS.DECRYPT (see Section 7.2.3).

9.3.3 KTS-KEM-KWS-basic

KTS-KEM-KWS-basic is the basic key transport scheme in the KTS-KEM-KWS family without key confirmation.

Let ($PubKey_V$, $PrivKey_V$) be party V's key-establishment key pair. Let K be the keying material to be transported from party U to party V. The parties **shall** perform the following or an equivalent sequence of steps, which are also illustrated in Figure 16.

Party U **shall** execute the following steps in order to transport keying material to party V.

Party U Actions:

1. Using party V's public key-establishment key $PubKey_V$, the length *kwkBits* of key to be used for key-wrapping, keying material K, and the additional input A, generate a ciphertext C (see Section 7.2.3.2), which includes an encrypted KWK as C_0 and the wrapped keying material as C_1:

   C = RSA-KEM-KWS.ENCRYPT($PubKey_V$, *kwkBits*, K, A).

2. Send the ciphertext C to party V.

Party V **shall** execute the following steps when receiving keys transported from party V.

Party V Actions:

1. Receive the transported keying material.

2. Using the ciphertext C, the private key-establishment key $PrivKey_V$, the length *kwkBits* of the key-wrapping key, and the additional input A, recover the keying material K (see Section 7.2.3.3):

   K = RSA-KEM-KWS.DECRYPT($PrivKey_V$, C, *kwkBits*, A).

   If the decryption operation outputs an error indicator, output an error indication and stop.

Output: The byte string K or an error indicator.

Sender (Party U)		**Receiver (Party V)**
K to be transported		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ← — — —	
C = RSA-KEM-KWS. ENCRYPT($PubKey_V$, *K*, *A*)	*C* ——————→	*K* = RSA-KEM-KWS. DECRYPT($PrivKey_V$, *C*, *A*)

Figure 16: KTS-KEM-KES-basic Scheme

9.3.4 KTS-KEM-KWS Key Confirmation

The **KTS-KEM-KWS-receiver-confirmation** scheme offers receiver confirmation and is based on the **KTS-KEM-KWS-basic scheme**.

9.3.4.1 KTS-KEM-KWS Common Components for Key Confirmation

The components for **KTS-KEM-KWS-receiver-confirmation** are the same as for **KTS-KEM-KWS-basic** (see Section 9.3.2), plus the following:

2. MAC: A message authentication code algorithm (see Section 5.2).

    a. the *MacKeyLen*: length in bytes of the *MacKey* (see Table 1 in Section 6.2.3).

    b. the *MacTagLen*: length in bytes of the *MacTag* (see Table 1 in Section 6.2.3).

For this scheme, the length of the keying material **shall** be at least *MacKeyLen,* where *MacKeyLen* is the length of the *MacKey* (see Section 6.2.3) and usually longer so that other keying material is available for subsequent operations. The *MacKey* **shall** be the first *MacKeyLen* bytes of the keying material and **shall** be used only for key confirmation.

9.3.4.2 KTS-KEM-KWS-receiver-confirmation

KTS-KEM-KWS-receiver-confirmation is a variant of **KTS-KEM-KWS-basic** with unilateral key confirmation from party V to party U.

Figure 17 depicts a typical flow for the **KTS-KEM-KWS-receiver-confirmation** scheme. In this scheme, party V, the receiver, and party U, the sender, assume the roles of key confirmation provider and recipient, respectively.

To provide (and receive) key confirmation (as described in Section 6.6.1), both parties set $EphemData_V = Null$, and $EphemData_U = C$:

Party V provides $MacTag_V$ to party U (as specified in Section 6.6.1, with P = V and R = U), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$$MacData_V = \text{“KC\_1\_V”} \| ID_V \| ID_U \| Null \| C\{ \| Text\}.$$

Party U uses the identical format and values to compute $MacTag_V$ and then verifies that the newly computed $MacTag_V$ matches the $MacTag_V$ value provided by party V.

The *MacKey* used during Key Confirmation **shall** be zeroized by Party V immediately after the computation of $MagTag_V$, and by Party U immediately after the verification of the received $MacTag_V$ or a (final) determination that the received $MacTag_V$ is in error.

Sender (Party U)		**Receiver (Party V)**
K = *MacKey* ll *KeyData*		($PubKey_V$, $PrivKey_V$)
Obtain party V's public key-establishment key	$PubKey_V$ ← — — —	
C = RSA-KEM-KWS. ENCRYPT($PubKey_V$, K, A)	C ——→	K = RSA-KEM.KWS. DECRYPT($PrivKey_V$, C, A)
		MacKey || *KeyData* = K
$MacTag_V$ =? MAC(*MacKey*, *MacTagLen*, $MacData_V$)	$MacTag_V$ ←——	$MacTag_V$ = MAC(*MacKey*, *MacTagLen*, $MacData_V$)

Figure 17: KTS-KEM-KWS-receiver-confirmation Scheme

9.3.5 KTS-KEM-KWS Security Properties

In each scheme included in this family, only the identifier of V (the receiver) is bound to a public key-establishment key. U (the sender) has assurance that no unintended party can recover the shared secret Z, and hence the KWK, from the ciphertext component C_0, and then use KWK to obtain the keying material K from the ciphertext component C_1 (without the compromise of some private information).

The receiver, however, has no such assurance. In particular, V has no assurance as to the accuracy of the identifier claimed by the sender and, therefore, has no assurance as to the true source of the ciphertext $C = C_0 \| C_1$ (or the transported K).

Due to the sender's unilateral selection of K, U can obtain assurance that fresh keying material has been transported. V has no such assurance.

A compromise of the receiver's private key will allow an adversary to masquerade as the receiver in future key establishment transactions, and compromises all keying material transported to the receiver in both past and future transactions.

In the key confirmation case, if one assumes that the transported keying material includes a *MacKey* that is (as required by this Recommendation) randomly generated for the particular **KTS-KEM-KWS-receiver-confirmation** transaction between U and V, then by successfully comparing the received value of $MacTag_V$ with its own computation, the sender (U) can obtain these assurances:

1. V has correctly recovered *Z*, and hence the KWK;
2. both parties agree on the values of ID_V, ID_U, and *C*;
3. the receiver (V) has correctly recovered (at least the *MacKey* portion of) *K* from *C*;
4. V possesses the correct value of the private key corresponding to the public key used in the transaction; and
5. V has actively participated in the transaction.

The use of a transaction-specific (random) *Z* (and hence the transaction-specific KWK) provides assurance to U that both C_0 and C_1 are also random for a given transaction, even if *K* is not. However, if U has transported the same *K* to multiple parties, and/or (in violation of the Recommendation) U has re-used a *MacKey*, then the return of a correct $MacTag_V$ value to U does not provide assurance that V has correctly obtained the keying material (or anything else). Anyone in possession of (at least the *MacKey* portion of) *K* could have computed $MacTag_V$.

10 Key Recovery

For some applications, the secret keying material used to protect data or to process protected data may need to be recovered (for example, if the normal reference copy of the secret keying material is lost or corrupted). In this case, either the secret keying material or sufficient information to reconstruct the secret keying material needs to be available (for example, the keys and other inputs to the scheme used to perform the key establishment process).

For example, the following information that is used during key establishment may need to be saved:

1. One or both keys of a key pair, as needed.
2. The nonce(s),
3. The ciphertext.
4. Additional input.
5. OtherInfo,
6. A symmetric key.

General guidance on key recovery and the protections required for each type of key is provided in the Recommendation for Key Management [7].

11. Implementation Validation

When the NIST Cryptographic Algorithm Validation System (CAVS) has established a validation program for this Recommendation, a vendor **shall** have its implementation tested and validated by the CMVP in order to claim conformance to this Recommendation. Information on the CMVP is available at http://csrc.nist.gov/cryptval/.

An implementation claiming conformance to this Recommendation **shall** include one or more of the following capabilities:

1. Key pair generation as specified in Section 6.3.
2. Explicit public key validation as specified in Section 6.4.3.
3. A key agreement scheme from Section 8, together with an **approved** key derivation function from Section 5.9. Other key derivation methods with specific protocols may be temporarily allowed for backward compatibility if agreed upon by the participating entities (i.e., party U and party V). These other allowable methods and the protocols that they may be used with are referenced in FIPS 140-2 Annex D. Documentation **shall** include how assurance of private key possession and assurance of public key validity are expected to be achieved by both the owner and the recipient.
4. A key transport scheme as specified in Section 9, together with an **approved** random bit generator, an **approved** hash function, an **approved** symmetric key-wrapping algorithm, and an **approved** key derivation function from Section 5.9 for RSA-KEM-KWS based schemes. Other key derivation methods with specific protocols may be temporarily allowed for backward compatibility if agreed upon by the participating entities (i.e., party U and party V). These other allowable methods and protocols are referenced in FIPS 140-2 Annex D.

An implementer **shall** also identify the appropriate specifics of the implementation, including:

1. The security strength(s) of supported cryptographic algorithms; this will determine the parameter set requirements (see Table 1 in Section 6.2.3),
2. The hash function to be used (see Section 5.1),
3. The *MacKey* length(s) (see Table 1 in Section 6.2.3),
4. The *MacTag* length (see Table 1 in Section 6.2.3),
5. The key establishment schemes available (see Sections 8 and 9),
6. The key derivation function to be used if a key agreement scheme is implemented, including the format of *OtherInfo* (see Section 5.9),
7. The type of nonces to be generated (see Section 5.6).
8. How assurance of private key possession and assurance of public key validity are expected to be achieved by both the owner and the recipient.

9. If a key transport scheme is implemented, indicate whether a capability is available to handle additional input.

Appendix A: Summary of Differences between this Recommendation and ANS X9.44 (Informative)

This list is informational and not meant to be exhaustive, but is intended to summarize important differences between this Recommendation and ANS X9.44. In general, this Recommendation can be seen as being more restrictive than ANS X9.44. The list of differences is as follows:

1. For purposes of validating an implementation of the schemes in this Recommendation durring an implementation validation test (under the NIST Cryptographic Algorithm Validation System), the value of *MacData* is set to the string "Standard Test Message", followed by a 128-bit field for a nonce. The default value for this field is all binary zeros. Different values for this field will be specified during testing. This is for the purpose of testing when no key confirmation capability exists. ANS X9.44 does not address implementation validation at this level of detail.
2. ANS X9.44 allows the public key exponent *e* to be as small as 3, whereas this Recommendation requires that *e* be at least 65537.
3. ANS X9.44 requires that separate keys be used for key transport and key agreement. This Recommendation allows the same key to be used for both purposes.
4. Regarding the key derivation function (KDF):
    a. This Recommendation specifies two **approved** KDFs, the concatenation KDF specified in Section 5.9.1 and the ASN.1 KDF specified in Section 5.9.2. Additional KDFs may be allowed for a transition period.
    b. ANS X9.44 provides two forms of a concatenation KDFs, KDF2 and KDF3. KDF2 is compatible with the concatenation KDFs specified in IEEE 1363[13], IEEE 1363a [14], ANS X9.42 [9], and ANS X9.63[11]. KDF3 can be used in a mode that is compatible with this Recommendation. The significant difference between KDF2 and KDF3 is that in KDF2, the counter is hashed after the shared secret, whereas in KDF3, the counter is hashed before the shared secret.
    c. The **approved** KDFs in this Recommendation require the input of the identifiers of the communicating parties; such information is allowed, but not required, in ANS X9.44.
    d. In this Recommendation, the shared secret is zeroized after a single call to a key derivation function, before the key agreement scheme releases any portion of the *DerivedKeyingMaterial* for use by relying applications.. The intent in ANS X9.44 is to prohibit the re-use of the shared secret, but the zeroization requirement is not specifically stated. An implication of this Recommendation's requirement concerning zeroization is that all of the keying material directly derived from the shared secret must be computed during one call to the KDF.

5. ANS X9.44 requires the use of X9-**approved** key-wrapping algorithms, whereas this Recommendation requires the use of NIST-**approved/allowed** key-wrapping algorithms. ASC X9 allows both AES and TDES key wrap algorithms [12], whereas NIST currently specifies only AES for key wrapping [8].
6. This Recommendation uses a more stringent definition of key confirmation than does ANS X9.44. ANS X9.44 does not require that the confirmation provider be authenticated. Therefore, schemes that qualify as offering key confirmation under ANS X9.44 may not qualify as offering key confirmation under this Recommendation. For example, the **kas1-bilateral-confirmation** scheme of ANS X9.44 does not exist in this Recommendation, since the identity of the initiator is not authenticated.
7. The KAS2 schemes are not provided in ANS X9.44. However, they are included in this Recommendation to provide schemes that are similar to the C(1,1) schemes in SP 800-56A.

Appendix B: Data Conversions (Normative)

B.1 Integer-to-Byte String (I2BS) Conversion

Input: A non-negative integer C and the intended length n of the byte string satisfying

$$2^{8n} > C$$

Output: A byte string S of length n bytes.

1. Let $S_1, S_2, \ldots, S_n$ be the bytes of S from leftmost to rightmost.
2. The bytes of S **shall** satisfy:

$$C = \Sigma 2^{8(n-i)} S_i \text{ for } i = 1 \text{ to } n.$$

B.2 Byte String to Integer (BS2I) Conversion

Input: A byte string S ($SLen$ is used to denote the length of the byte string).

Output: A non-negative integer C.

Steps:

1. Let $S_1\ S_2\ \ldots\ S_{sLen}$ be the bytes of S from first to last, and let x_{SLen-i} be the integer value of the octet x_i for $1 \le i \le SLen$, where the integer value is represented as a byte (i.e., an eight-bit string), with the most significant bit first (i.e., on the left).
2. Let $x = x_{SLen-1} \cdot 256^{SLen-1} + x_{SLen-2} \cdot 256^{SLen-2} + \ldots + x_1 \cdot 256 + x_0$.
3. Output x.

Appendix C: Prime Factor Recovery (Normative)

The following algorithm recovers the prime factors of a modulus, given the public and private exponents. The algorithm is based on Fact 1 in [18].

Function call: RecoverPrimeFactors(n, e, d)

Input:

1. n: modulus
2. e: public exponent
3. d: private exponent

Output:

1. (p, q): prime factors of modulus

Errors: "prime factors not found"

Assumptions: The modulus n is the product of two prime factors p and q; the public and private exponents satisfy $d \times e \equiv 1 \pmod{\lambda(n)}$ where $\lambda(n) = \text{LCM}(p-1, q-1)$

Process:

1. Let $k = d \times e - 1$. If k is odd, then go to Step 4.
2. Write k as $k = r\,2^t$, where r is the largest odd integer dividing k, and $t \geq 1$.
3. For $i = 1$ to 100 do:
    a. Generate a random integer g in the range $[0, n-1]$.
    b. Let $y = g^r \bmod n$.
    c. If $y = 1$ or $y = n - 1$, then go to Step g.
    d. For $j = 1$ to $t - 1$ do:
        I. Let $x = y^2 \bmod n$.
        II. If $x = 1$, go to Step 5
        III. If $x = n - 1$, go to Step3.7.
        IV. Let $y = x$.

e. Let $x = y^2 \bmod n$.

f. If $x = 1$, go to Step 5.

g. Continue.

4. Output "prime factors not found" and stop.
5. Let $p = \text{GCD}(y - 1, n)$ and let $q = n/p$.
6. Output (p, q) as the prime factors.

Notes:

1. According to Fact 1 in [18], the probability that one of the values of y in an iteration of Step 3 reveals the factors of the modulus is at least 1/2, so on average, at most two iterations of that step will be required. If the prime factors are not revealed after 100 iterations, then the probability is overwhelming that the modulus is not the product of two prime factors, or that the public and private exponents are not consistent with each other.
2. The algorithm bears some resemblance to the Miller-Rabin primality testing algorithm (see, e.g., ANS X9.80).
3. The order of the recovered prime factors p and q may be the reverse of the order in which the factors were generated originally.

Appendix D: References (Informative)

[1] FIPS 140-2, Security requirements for Cryptographic Modules, May 25, 2001. FIPS 140-3 is currently under development.

[2] FIPS 180-3, Secure Hash Standard, June 2007 (Draft).

[3] FIPS 186-3, Digital Signature Standard, March 2006 (Draft)

[4] FIPS 197, Advanced Encryption Standard, November 2001.

[5] FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008.

[6] NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005.

[7] NIST SP 800-57-Part 1, Recommendation for Key Management, August 2005.

[8] AES Key Wrap Specification, NIST, November 16, 2001.

[9] ANS X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[10] [ANS X9.44 Public Key Cryptography for the Financial Services Industry: Key Establishment Using Integer Factorization Cryptography, August 24, 2007.

[11] ANS X9.63-2001, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

[12] ANSI X9.102-2008, Symmetric Key Cryptography for the Financial Services Industry – Wrapping of Keys and Associated Data.

[13] IEEE 1363-2000, IEEE Standard Specifications for Public Key Cryptography.

[14] IEEE 1363a-2004, IEEE Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques.

[15] *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0*, J. Manger, In J. Kilian, editor, Advances in Cryptology – Crypto 2001, pp. 230 – 238, Springer Verlag, 2001.

[16] *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* R. Rivest, A. Shamir and L. Adleman, Communications of the ACM, 21(2), pp. 120 – 126, February 1978.

[17] *The Security of all RSA and Discrete Log Bits,* J. Håstad and M. Näslund. Proc. of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 510 – 521, 1998.

[18] *Twenty Years of Attacks on the RSA Cryptosystem*, D. Boneh, Notices of the American Mathematical Society (AMS), 46(2), 203 – 213. 1999.