# IEEE P1363 / D13 (Draft Version 13)

# Standard Specifications for Public Key Cryptography

**Abstract.** This standard specifies common public-key cryptographic techniques, including mathematical primitives for secret value (key) derivation, public-key encryption, and digital signatures, and cryptographic schemes based on those primitives. It also specifies related cryptographic parameters, public keys and private keys. The purpose of this standard is to provide a reference for specifications of a variety of techniques from which applications may select.

**Keywords.** Public-key cryptography; key agreement; digital signature; encryption.



Comments and suggestions are welcome. Please contact the chair, Ari Singer, at singerar@pb.com.

## Introduction

(This introduction is not part of IEEE P1363, Standards Specifications for Public-Key Cryptography).

The P1363 project started as the "Standard for Rivest-Shamir-Adleman, Diffie-Hellman, and Related Public-Key Cryptography" with its first meeting in January 1994, following a strategic initiative by the Microprocessor Standards Committee to develop standards for cryptography.

P1363's scope broadened with the inclusion of elliptic curve cryptosystems as "related" cryptography, and later the title was changed to the current one to reflect the breadth of the effort. In mid-1996, the working group decided to include three families of techniques, based on three different hard problems—integer factorization, discrete logarithms over finite fields, and elliptic curve discrete logarithms. By late 1996, the set of techniques was fairly stable, with "additional" techniques deferred to the P1363a project.

Most of the next two years saw an increasing intensity of editing as the working group sought to fulfill the scope that remained. In early 1998, the group set a schedule for completion that would bring the document to ballot in late 1998, and final sections of the document were prepared.

The process of developing a standard is always a challenging one, particularly when the subject is as technical as cryptography and the scope is as broad as proposed for P1363. Moreover, as other groups were developing complementary standards at the same time as P1363, close coordination was an essential aspect. Security implies a great deal of caution, and the working group was careful in its deliberations not to set any "standards" that might later lead to vulnerabilities (although, as it is pointed out elsewhere, this standard is much more about a framework for specifying public-key techniques, than it is about security per se).

In addition to this standard, the P1363 project has provided a number of other contributions to the computer security industry. First, it has presented a forum where experts can discuss general issues related to public-key standardization. Second, through its Web page and its call for submissions to P1363a, the project has given a focal point for the presentation of new developments in public-key technology. Third, it has helped facilitate the open discussion of intellectual property issues in this area. And, finally, through its drafts, P1363 has provided reference material to a wide community of cryptographers that otherwise was relegated to textbooks and research papers. For the duration of its existence, the working group intends to maintain a web page containing “errata and latest information” as an additional reference to support P1363 documents (see http://grouper.ieee.org/groups/1363/index.html).

The P1363 working group is grateful to the many experts who have contributed to the standard, and particularly to those whose development of public-key technology over the past two decades has provided the foundation for information security in the next century.

The active participants in the P1363 working group at the time this document was completed and balloted were as follows:

Benjamin Arazi
Terry S. Arnold, Vice Chair (1994-)
Ian Blake
Lily Chen
Louis Finkelstein
Walter Fumy
Don B. Johnson
Tatsuaki Okamoto
Minghua Qu
Anand Rajan
Leonid Reyzin
Allen Roginsky
Roger Schlafly, Secretary (1994-)
Richard Schroeppel

Burt Kaliski, Chair (1994-)
Shirley Kawamoto
David Kravitz
Pil Joong Lee
Franck Leprevost
Michael Markowitz, Treasurer (1996-)
Alfred Menezes
Ari Singer
Jerry Solinas
Kazuo Takaragi
Ashok Vadekar
Scott Vanstone
Yiqun Lisa Yin, Editor (1996-)
Robert Zuccherato

In addition, the working group would like to thank the following people for their contributions to the standard:

Michel Abdalla
Rich Ankney
David Aucsmith, Editor (1994-95)
Paulo S. L. M. Barreto
Mihir Bellare
Tom Berson
Simon Blake-Wilson
Eric Blossom
Uri Blumenthal
Mark Chen
Richard Crandall
Wei Dai
Erik De Win
Jean-Francois Dhem
Whitfield Diffie
Carl Ellison
Amos Fiat
Robert Gallant
John Gilmore
Roger Golliver
Gary Graunke
Phillip Griffin
Louis Guillou
Stuart Haber
Shouichi Hirose
Robert Hofer
Dale Hopkins
David Hopwood
Russell Housley
Eric Hughes
David Jablon
Aleksandar Jurisic
John Kennedy, Treasurer (1996)
Katherine T. Kislitzin
Cetin K. Koc
Ray Kopsa
Robert J. Lambert
Peter Landrock
Laurie Law
Chang-Hyi Lee
Jong-In Lim
Moses Liskov
Wenbo Mao
Stephen M. Matyas
Preda Mihailescu
Peter Montgomery
Francois Morain
Peter Neumann
Mark Oliver, Editor (1994-96)
Aram Perez
Mohammad Peyravian
Bart Preneel
Jean-Jacques Quisquater
Karen Randall
Richard L. Robertson
Matt Robshaw
Phillip Rogaway
Paul Rubin
Rainer Rueppel
Claus P Schnorr
Mike Scott
Gadiel Seroussi
Sherry Shannon
Robert D. Silverman
Tim Skorick
David Sowinski
Paul Van Oorschot
Michael J. Wiener
Harold M. Wilensky
Thomas Wu
Susumu Yoshida
Yuliang Zheng

The working group apologizes for any inadvertent omissions from the above list. Please note that inclusion of a person's name on the above two lists does not imply that the person agrees with all the materials in the standard.

## Contents

# 1. Overview

## 1.1 Scope

Specifications of common public-key cryptographic techniques, including mathematical primitives for secret value (key) derivation, public-key encryption, and digital signatures, and cryptographic schemes based on those primitives. Specifications of related cryptographic parameters, public keys and private keys. Class of computer and communications systems is not restricted.

NOTE—As of the date of this standard's publication, another IEEE project, P1363a, is underway to specify additional techniques. Its intent is to be supplemental to this standard. Its scope is similar to that of this standard, with the addition of identification schemes. For the purpose of P1363a, see the note in Section 1.2.

## 1.2 Purpose

The transition from paper to electronic media brings with it the need for electronic privacy and authenticity. Public-key cryptography offers fundamental technology addressing this need. Many alternative public-key techniques have been proposed, each with its own benefits. However, there has been no single, comprehensive reference defining a full range of common public-key techniques covering key agreement, public-key encryption, digital signatures, and identification from several families, such as discrete logarithms, integer factorization, and elliptic curves.

It is not the purpose of this project to mandate any particular set of public-key techniques, or particular attributes of public-key techniques such as key sizes. Rather, the purpose is to provide a reference for specifications of a variety of techniques from which applications may select.

NOTE—As of the date of this standard's publication, another IEEE project, P1363a, is underway to specify additional techniques. Its intent is to be supplemental to this standard. Its purpose is similar to that of this standard, with the difference that P1363a will focus on newer techniques while this standard specifies relatively well-established techniques. For the scope of P1363a, see the note in Section 1.1.

## 1.3 Organization of the Document

This standard contains two parts: the main document and annexes.

### 1.3.1 Structure of the Main Document

— Section 1 is a general overview.
— Section 2 provides references to other standards and publications.
— Section 3 defines some terms used throughout this standard.
— Section 4 gives an overview of the types of cryptographic techniques that are defined in this standard.
— Section 5 describes certain mathematical conventions used in the standard, including notation and representation of mathematical objects. It also defines formats to be used in communicating the mathematical objects as well as primitives for data type conversion.
— Sections 6, 7, 8, 9, 10 and 11 define three families of cryptographic techniques: techniques based on the discrete logarithm problem (DL), elliptic curve discrete logarithm problem (EC), and integer factorization problem (IF). Sections 6, 7 and 8 define the setting and primitives used in the DL, EC

and IF families, respectively. Sections 9, 10 and 11 define key agreement schemes, signature schemes and encryption schemes, respectively.

— Section 12 defines message encoding methods for signature and encryption schemes in Sections 10 and 11.
— Section 13 defines key derivation functions for key agreement schemes in Section 9.
— Section 14 defines certain auxiliary functions supporting the techniques in Sections 12 and 13.

### 1.3.2 Structure of the Annexes

The annexes provide background and helpful information for the users of the standard and include the following sections.

| | |
|---|---|
| Annex A (Informative) | Number-Theoretic Background |
| Annex B (Normative) | Conformance |
| Annex C (Informative) | Rationale |
| Annex D (Informative) | Security Considerations |
| Annex E (Informative) | Formats |
| Annex F (Informative) | Bibliography |

## 2. References

This standard shall be used in conjunction with following publications:

FIPS 180-1, *Secure Hash Standard*, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, Virginia, April 17, 1995 (supersedes FIPS PUB 180). Available at `http://www.itl.nist.gov/div897/pubs/fip180-1.htm`.

ISO/IEC 10118-3:1998 Information Technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.

Notes

1—The above references are required for implementing some of the techniques in this document, but not all the techniques.

2—The mention of any standard in this document is for reference only, and does not imply conformance with that standard. Readers should refer to the relevant standard for full information on conformance with that standard.

3—Bibliography is provided in Annex F.

## 3. Definitions

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

IEEE Std 100-1996 should be referenced for terms not defined in this clause.

**authentication of ownership:** the assurance that a given, identified party intends to be associated with a given public key. May also include assurance that the party possesses the corresponding private key (see D.3.2. for more information).

**bit length:** See: length.

**bit string:** an ordered sequence of bits (0's and 1's). A bit and a bit string of length 1 are equivalent for all purposes of this standard.

**ciphertext:** the result of applying encryption to a message. Contrast: plaintext. See also: encryption.

**conformance region:** a set of inputs to a primitive or a scheme operation for which an implementation operates in accordance with the specification of the primitive or scheme operation (see B.1 for more information).

**cryptographic family:** there are three families of techniques presented in this standard, based on the underlying hard problem: discrete logarithm over finite fields (DL), discrete logarithm over elliptic curve groups (EC), and integer factorization (IF).

**decrypt:** to produce plaintext from ciphertext. Contrast: encrypt. See also: ciphertext; encryption; plaintext.

**digital signature:** a digital string for providing authentication. Commonly, in public-key cryptography, it is a digital string that binds a public key to a message in the following way: only the person knowing the message and the corresponding private key can produce the string, and anyone knowing the message and the public key can verify that the string was properly produced. A digital signature may or may not contain the information necessary to recover the message itself. See also: digital signature scheme: public key; public-key cryptography; private key; signature scheme with appendix; signature scheme with message recovery.

**digital signature scheme:** a method for providing authentication. In public-key cryptography, this method can be used to generate a digital signature on a message with a private key in such a way that anyone knowing the corresponding public key can verify that the digital signature was properly produced. See also: digital signature; public key; public-key cryptography; private key; signature scheme with appendix; signature scheme with message recovery.

**domain parameters:** a set of mathematical objects, such as fields or groups, and other information, defining the context in which public/private key pairs exist. More than one key pair may share the same domain parameters. Not all cryptographic families have domain parameters. See also: public/private key pair; valid domain parameters.

**domain parameter validation:** the process of ensuring or verifying that a set of domain parameters is valid. See also: domain parameters; key validation; valid domain parameters.

**encrypt:** to produce ciphertext from plaintext. Contrast: decrypt. See also: ciphertext; encryption; plaintext.

**encryption scheme:** a method for providing privacy. In public-key cryptography, this method can be used to modify a message with the help of a public key to produce what is known as ciphertext in such a way that only the holder of the corresponding private key can recover the original message from the ciphertext. See also: ciphertext; plaintext; private key; public key; public-key cryptography.

**family:** See: cryptographic family.

**field**: a setting in which the usual mathematical operations (addition, subtraction, multiplication, and division by nonzero quantities) are possible and obey the usual rules (such as the commutative, associative, and distributive laws). A DL or EC scheme is always based on computations in a field. See [Kob94] for a precise mathematical definition.

**finite field**: a field in which there are only a finite number of quantities. The DL and EC schemes are always implemented over finite fields. See Section 5 for a description of the particular finite fields used in this standard.

**first bit:** the leading bit of a bit string or an octet. For example, the first bit of 0110111 is 0. Contrast: last bit. Syn: most significant bit; leftmost bit. See also: bit string; octet.

**first octet:** the leading octet of an octet string. For example, the first octet of 1c 76 3b e4 is 1c. Contrast: last octet. Syn: most significant octet; leftmost octet. See also: octet; octet string.

**key agreement:** a method by which two entities, using each other's public keys and their own private keys, agree on a common secret key that only they know. The secret key is then commonly used in some symmetric cryptography technique. See also: private key; public key; secret key; symmetric cryptography.

**key confirmation:** the assurance provided to each party participating in a key agreement protocol that the other party is capable of computing the agreed-upon key, and that it is the same for both parties (see D.5.1.3 for more information). See also: key agreement.

**key derivation:** the process of deriving a secret key from a secret value. See also: secret key; secret value.

**key pair:** See: public/private key pair.

**key validation:** the process of ensuring or verifying that a key or a key pair is valid. See also: domain parameter validation; public/private key pair; valid key; valid key pair.

**last bit:** The trailing bit of a bit string or an octet. For example, the last bit of 0110111 is 1. Contrast: first bit. Syn: least significant bit; rightmost bit. See also: first bit; octet.

**last octet:** The trailing octet of an octet string. For example, the last octet of 1c 76 3b e4 is e4. Contrast: first octet. Syn: least significant octet; rightmost octet. See also: octet; octet string.

**least significant:** See: last bit; last octet.

**length:** (1) Length of a bit string is the number of bits in the string. (2) Length of an octet string is the number of octets in the string. (3) Length in bits of a nonnegative integer $n$ is $\lceil \log_2 (n + 1) \rceil$ (i.e., the number of bits in the integer's binary representation). (4) Length in octets of a nonnegative integer $n$ is

$\lceil \log_{256} (n + 1) \rceil$ (i.e., the number of digits in the integer's representation base 256). For example, the length in bits of the integer 500 is 9, whereas its length in octets is 2.

**leftmost bit:** See: first bit.

**leftmost octet:** See: first octet.

**message recovery:** See: signature with message recovery.

**message representative:** a mathematical value for use in a cryptographic primitive, computed from a message that is input to an encryption or a digital signature scheme. See also: encryption scheme; digital signature scheme.

**most significant:** See: first bit; first octet.

**octet:** A bit string of length 8. An octet has an integer value between 0 and 255 when interpreted as a representation of an integer in base 2. An octet can also be represented by a hexadecimal string of length 2, where the hexadecimal string is the representation of its integer value base 16. For example, the integer value of the octet 10011101 is 157; its hexadecimal representation is 9d. See also: bit string.

**octet string:** An ordered sequence of octets. See also: octet.

**parameters:** See: domain parameters.

**plaintext:** a message before encryption has been applied to it; the opposite of ciphertext. Contrast: ciphertext. See also: encryption.

**private key:** the private element of the public/private key pair. See also: public/private key pair; valid key.

**public key:** the public element of the public/private key pair. See also: public/private key pair; valid key.

**public-key cryptography:** methods that allow parties to communicate securely without having prior shared secrets, usually through the use of public/private key pairs. Contrast: symmetric cryptography. See also: public/private key pair.

**public/private key pair:** a pair of cryptographic keys used in public-key cryptography, consisting of a public key and a private key that correspond to each other by some mathematical relation. The public key is commonly available to a wide audience and can be used to encrypt messages or verify digital signatures; the private key is held by one entity and not revealed to anyone--it is used to decrypt messages encrypted with the public key and/or produce signatures that can verified with the public key. A public/private key pair can also be used in key agreement. In some cases, a public/private key pair can only exist in the context of domain parameters. See also: digital signature; domain parameters; encryption; key agreement; public-key cryptography; valid key; valid key pair.

**rightmost bit:** See: last bit.

**rightmost octet:** See: last octet.

**root:** if $f(x)$ is a polynomial, then its root is a value $r$ of the variable $x$ such that $f(r) = 0$.

**secret key:** a key used in symmetric cryptography; commonly needs to be known to all parties involved, but cannot be known to an adversary. Contrast: public/private key pair. See also: key agreement; shared secret key; symmetric cryptography.

**secret value:** a value that can be used to derive a secret key, but typically cannot by itself be used as a secret key. See also: secret key.

**shared secret key:** a secret key shared by two parties, usually derived as a result of a key agreement scheme. See also: key agreement; secret key.

**shared secret value:** a secret value shared by two parties, usually during a key agreement scheme. See also: key agreement; secret value.

**signature:** See: digital signature.

**signature scheme with appendix:** a digital signature scheme that requires the signed message as input to the verification algorithm. Contrast: signature scheme with message recovery. See also: digital signature; digital signature scheme.

**signature scheme with message recovery:** a digital signature scheme that contains enough information for recovery of the signed message, thus limiting the possible message size while eliminating the need to transmit the message with the signature and input it to the verification algorithm. Contrast: signature scheme with appendix. See also: digital signature; digital signature scheme.

**signature verification:** the process of verifying a digital signature. See also: digital signature; digital signature scheme.

**symmetric cryptography:** methods that allow parties to communicate securely only when they already share some prior secrets, such as the secret key. Contrast: public-key cryptography. See also: secret key.

**valid domain parameters:** a set of domain parameters that satisfies the specific mathematical definition for the set of domain parameters of its family. While a set of mathematical objects may have the general structure of a set of domain parameters, it may not actually satisfy the definition (for example, it may be internally inconsistent) and thus not be valid. See also: domain parameters; public/private key pair; valid key; valid key pair; validation.

**valid key:** a key (public or private) that satisfies the specific mathematical definition for the keys of its family, possibly in the context of its set of domain parameters. While some mathematical objects may have the general structure of keys, they may not actually lie in the appropriate set (for example, they may not lie in the appropriate subgroup of a group or be out of the bounds allowed by the domain parameters) and thus not be valid keys. See also: domain parameters; public/private key pair; valid domain parameters; valid key pair; validation.

**valid key pair**: a public/private key pair that satisfies the specific mathematical definition for the key pairs of its family, possibly in the context of its set of domain parameters. While a pair of mathematical objects may have the general structure of a key pair, the keys may not actually lie in the appropriate sets (for example, they may not lie in the appropriate subgroup of a group or be out of the bounds allowed by the domain parameters) or may not correspond to each other; such a pair is thus not a valid key pair. See also: domain parameters; public/private key pair; valid domain parameters; valid key; validation.

**validation:** See: domain parameter validation; key validation.

# 4. Types of Cryptographic Techniques

This section gives an overview of the types of cryptographic techniques that are specified in this standard as well as some requirements for conformance with those techniques. See Annex B for more on conformance.

## 4.1 General Model

As stated in Section 1, the purpose of this standard is to provide a reference for specifications of a variety of common public-key cryptographic techniques from which applications may select. Therefore, it is desirable to define these techniques in a framework that would allow selection of techniques appropriate for particular applications. Different types of cryptographic techniques can be viewed abstractly according to the following three-level general model.

— *Primitives* – basic mathematical operations. Historically, they were discovered based on number-theoretic hard problems. Primitives are not meant to achieve security just by themselves, but they serve as building blocks for schemes.

— *Schemes* – a collection of related operations combining primitives and additional methods (Section 4.4). Schemes can provide complexity-theoretic security which is enhanced when they are appropriately applied in protocols.

— *Protocols* – sequences of operations to be performed by multiple parties to achieve some security goal. Protocols can achieve desired security for applications if implemented correctly.

From an implementation viewpoint, primitives can be viewed as low-level implementations (e.g., implemented within cryptographic accelerators, or software modules), schemes can be viewed as medium-level implementations (e.g., implemented within cryptographic service libraries), and protocols can be viewed as high-level implementations (e.g., implemented within entire sets of applications).

General frameworks for primitives, schemes, and additional methods are provided in Sections 4.2, 4.3, and 4.4, and specific techniques are defined in Sections 6 through 14. This standard, however, does not define protocols, mainly because they tend to be application-specific and hence are outside the scope of the standard. Nevertheless, the techniques defined in this standard are key components for constructing various cryptographic protocols. Also, Annex D discusses security considerations related to how the techniques can be used in protocols to achieve certain security attributes.

## 4.2 Primitives

The following types of primitives are defined in this standard:

— Secret Value Derivation Primitives (SVDP), components of key agreement schemes.

— Signature Primitives (SP) and Verification Primitives (VP), components of signature schemes.

— Encryption Primitives (EP) and Decryption Primitives (DP), components of encryption schemes.

Primitives in this standard are presented as mathematical operations, and they are not meant to provide security by themselves. For example, it may be easy to compute message-signature pairs that a signature

verification primitive would find consistent with a public key. This is no assurance, however, that the signature was produced on the message by the owner of the corresponding private key. Such assurance can only be provided by a properly implemented signature scheme, which uses the primitive along with other operations.

Primitives assume that their inputs satisfy certain assumptions, as listed with the specification of each primitive. An implementation of a primitive is unconstrained on an input not satisfying the assumptions, as long as it does not adversely affect future operation of the implementation; the implementation may or may not return an error condition. For example, an implementation of a signature primitive may return something that looks like a signature even if its input was not a valid private key. It may also reject the input. It is up to the user of the primitive to guarantee that the input will satisfy the constraints or to include the relevant checks. For example, the user may choose to use the relevant key and domain parameter validation techniques.

The specification of a primitive consists of the following information:

— *input* to the primitive
— *assumptions* about the input made in the description of the operation performed by the primitive
— *output* from the primitive
— *operation* performed by the primitive, expressed as a series of *steps*
— *conformance region recommendations* describing the minimum recommended set of inputs for which an implementation should operate in conformance with the primitive (see Annex B for more on conformance)

The specifications are functional specifications, not interface specifications. As such, the format of inputs and outputs and the procedure by which an implementation primitive is invoked are outside the scope of this standard. See Annex E for more information on input and output formats.

## 4.3 Schemes

The following types of schemes are defined in this standard:

— Key Agreement Schemes (KAS), in which two parties use their public/private key pairs and possibly other information to agree on a shared secret key. The shared secret key may then be used for symmetric cryptography (see Section 3 for the definition of symmetric cryptography).

— Signature Schemes with Appendix (SSA), in which one party signs a message using its private key, and any other party can verify the signature by examining the message, the signature and the signer's corresponding public key.

— Encryption Schemes (ES), in which any party can encrypt a message using a recipient's public key, and only the recipient can decrypt the message by using its corresponding private key. Encryption schemes may be used for establishing on secret keys to be used in symmetric cryptography.

Schemes in this standard are presented in a general form based on certain primitives and additional methods, such as message encoding methods. For example, an encryption scheme is based on an encryption primitive, a decryption primitive, and an appropriate message encoding method.

Schemes also include key management operations, such as selecting a private key or obtaining another party's public key. For proper security, a party needs to be assured of the true owners of the keys and domain parameters and of their validity. Generation of domain parameters and keys needs to performed

properly, and in some cases validation also needs to be performed. While outside the scope of this standard, proper key management is essential for security. It is addressed in more detail in Annex D.

The specification of a scheme consists of the following information:

— *scheme options*, such as choices for primitives and additional methods
— one or more *operations*, depending on the scheme, expressed as a series of *steps*
— *conformance region recommendations* for implementations conforming with the scheme (see Annex B for more on conformance)

As for primitives, the specifications are functional specifications, not interface specifications. As such, the format of inputs and outputs and the procedure by which an implementation of a scheme is invoked are outside the scope of this standard. See Annex E for more information on input and output formats.

## 4.4 Additional Methods

This standard specifies the following additional methods:

— Message Encoding Methods, which are components of signature or encryption schemes.

  — Encoding Methods for Signatures with Appendix (EMSA)
  — Encoding Method for Encryption (EME)

— Key Derivation Functions (KDF), which are a component of key agreement schemes.

— Auxiliary Functions, which are building blocks for other additional methods.

  — Mask Generation Function (MGF), which is used in EME
  — Hash Functions, which are used in EMSA, EME, KDF, and MGF

The specified additional methods are strongly recommended for use in the schemes. The use of an inadequate message encoding method, key derivation function, or auxiliary function may compromise the security of the scheme in which it is used. Therefore, any implementation which chooses not to follow the recommended additional methods for a particular scheme should perform its own thorough security analysis of the resulting scheme.

## 4.5 Table Summary

This section gives a summary of all the schemes in this standard, together with the primitives and additional methods that are invoked within a scheme.

| Scheme Name | Primitives | Additional Methods |
|---|---|---|
| ***key agreement schemes*** | | |
| DLKAS-DH1 | DL/ECSVDP-DH or -DHC | KDF1 (uses a hash function) |
| DL/ECKAS-DH2 | DL/ECSVDP-DH and/or -DHC | KDF1 (uses a hash function) |
| DLKAS-MQV | DL/ECSVDP-MQV or -MQVC | KDF1 (uses a hash function) |
| ***signature schemes with appendix*** | | |
| DL/ECSSA | DLSP-NR and DLVP-NR<br>or<br>DLSP-DSA and DLVP-DSA<br>or<br>ECSP-NR and ECVP-NR<br>or<br>ECSP-DSA and ECVP-DSA | EMSA1 (uses a hash function) |
| IFSSA | IFSP-RSA1 and IFVP-RSA1<br>or<br>IFSP-RSA2 and IFVP-RSA2<br>or<br>IFSP-RW and IFVP-RW | EMSA2 (uses a hash function) |
| ***encryption schemes*** | | |
| IFES | IFEP-RSA and IFDP-RSA | EME1 (uses a hash function and a mask generation function) |

# 5. Mathematical Conventions

This section describes certain mathematical conventions used in the standard, including notation and representation of mathematical objects. It also contains primitives for data type conversion. Note that the internal representation of mathematical objects is left entirely to the implementation, and may or may not follow the one described below.

## 5.1 Mathematical Notation

The following mathematical notation is used throughout the document.

| | |
|---|---|
| 0 | Denotes the integer 0, the bit 0, or the additive identity (the element zero) of a finite field. See Section 5.3 for more on finite fields. |
| 1 | Denotes the integer 1, the bit 1, or the multiplicative identity (the element one) of a finite field. See Section 5.3 for more on finite fields. |
| $a \times b$ | The product of $a$ and $b$, where $a$ and $b$ are either both integers, or both finite field elements. When it does not cause confusion, $\times$ is omitted and the notation $ab$ is used. See Section 5.3 for more on finite fields. |
| $a \times P$ | Scalar multiplication of an elliptic curve point $P$ by a non-negative integer $a$. When it does not cause confusion, $\times$ is omitted and the notation $aP$ is used. See Section 5.4 for more on elliptic curves. |
| $\lceil x \rceil$ | The smallest integer greater than or equal to the real number $x$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$. |
| $\lfloor x \rfloor$ | The largest integer less than or equal to the real number $x$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$. |
| $[a, b]$ | The interval of integers between and including the integers $a$ and $b$. |
| LCM ($a$, $b$) | For two positive integers $a$ and $b$, denotes the *least common multiple* of $a$ and $b$, i.e., the least positive integer that is divisible by both $a$ and $b$. See Annex A.1.1 and A.2.2 for an algorithm to compute the LCM. |
| GCD ($a$, $b$) | For two positive integers $a$ and $b$, denotes the *greatest common divisor* of $a$ and $b$, i.e., the largest positive integer that divides both $a$ and $b$. See Annex A.1.1 and A.2.2 for an algorithm to compute the GCD. |
| $X \oplus Y$ | Bitwise exclusive-or (XOR) of two bit strings or two octet strings $X$ and $Y$ of the same length. |
| $X \parallel Y$ | Ordered concatenation of two strings $X$ and $Y$. $X$ and $Y$ are either both bit strings, or both octet strings. |
| $\log_2 x$ | The logarithmic function of a positive number $x$ to the base 2. |
| $\log_{256} x$ | The logarithmic function of a positive number $x$ to the base 256. |
| $a \bmod n$ | The unique remainder $r$, $0 \le r < n$, when the integer $a$ is divided by the positive integer $n$. For example, $23 \bmod 7 = 2$. The operator "mod" has the lowest precedence of all arithmetic operators (e.g., $5 + 8 \bmod 3$ is equal to $13 \bmod 3$, not $5 + 2$). See Annex A.1.1 for more details. |
| $a \equiv b \pmod n$ | The integers $a$ and $b$ have the same remainder when divided by the positive integer $n$. Pronounced "$a$ is congruent to $b$ modulo $n$." This is equivalent to $(a \bmod n) = (b \bmod n)$. See Annex A.1.1 for more details. |
| $a \not\equiv b \pmod n$ | The integers $a$ and $b$ have different remainders when divided by the positive integer $n$. Pronounced "$a$ is not congruent to $b$ modulo $n$." This is equivalent to $(a \bmod n) \neq (b \bmod n)$. |

| | |
|---|---|
| $a^{-1} \bmod n$ | A positive integer $b < n$, if it exists, such that $ab \bmod n = 1$. Prounouced "the (multiplicative) inverse of $a$ modulo $n$." Also denoted by $1/a \bmod n$. See Annex A.1.1 and A.2.2 for a more detailed description and an algorithm for computing it. |
| $a/b \bmod n$ | An integer $a$ multiplied by the inverse of the integer $b$ modulo $n$. Equivalent to $ab^{-1} \bmod n$. |
| $GF(p)$ | The finite field of $p$ elements, represented as the integers modulo $p$, where $p$ is an odd prime number. Also known as a *prime finite field*. See Section 5.3.1 for more information. |
| $GF(2^m)$ | The finite field containing $2^m$ elements for some integer $m > 0$. Also known as a *characteristic two finite field*. See Section 5.3.2 for more information. |
| $GF(q)$ | The finite field containing $q$ elements. For this standard, $q$ will be either a prime number ($p$) or a power of 2 ($2^m$). |
| $\left(\frac{x}{n}\right)$ | The Jacobi symbol of the integer $x$ with respect to the integer $n$. See Annex A.1.4 for a detailed description and A.2.3 for an algorithm to compute the Jacobi symbol. |
| 0 | The elliptic curve point at infinity. See Section 5.4 for more information. |
| $exp(a, b)$ | The result of raising $a$ to the power $b$, where $a$ is an integer or a finite field element, and $b$ is an integer. Also denoted by $a^b$. |
| $exp(a)$ | The result of raising the number $e$ (where $e = 2.718\ldots$) to the power $a$, where $a$ is a real number. |

NOTE—Throughout this main document, integers and field elements are denoted with lower-case letters; while octet strings and elliptic curve points are denoted with upper-case letters. The one exception is when a public key that can be either a field element or an elliptic curve point is denoted by a lower-case letter in the DL and EC schemes (Sections 9.2, 9.3, 9.4 and 10.2).

## 5.2 Bit Strings and Octet Strings

Bit strings and octet strings are ordered sequences. The terms "first" and "last," "leftmost" and "rightmost," and "leading" and "trailing" are used to distinguish the ends of these sequences ("first," "leftmost" and "leading" are equivalent; "last," "rightmost" and "trailing" are equivalent; other publications sometimes use "most significant," which is synonymous with "leading," and "least significant," which is synonymous with "trailing").

Note that when a string is represented as a sequence, it may be indexed from right to left or from left to right, starting with any index; this does not change the meaning of the terms above. For example, consider the octet string of 4 octets: 1c 76 3b e4. One can represent it as a string $a_0\ a_1\ a_2\ a_3$ with $a_0$ = 1c, $a_1$ = 76, $a_2$ = 3b, and $a_3$ = e4. In this case, $a_0$ represents the first octet, and $a_3$ represents the last octet. Alternatively, one can represent it as a string $a_1\ a_2\ a_3\ a_4$ with $a_1$ = 1c, $a_2$ = 76, $a_3$ = 3b, and $a_4$ = e4. In this case, $a_1$ represents the first octet, and $a_4$ represents the last octet. Yet another possibility would be to represent it as $a_3\ a_2\ a_1\ a_0$ with $a_3$ = 1c, $a_2$ = 76, $a_1$ = 3b, and $a_0$ = e4. In this case, $a_3$ represents the first octet and $a_0$ represents the last octet. No matter how this string is represented, the value of the first octet is always 1c and the value of the last octet is always e4.

## 5.3 Finite Fields

This section describes the kinds of underlying finite fields $GF(q)$ that shall be used, and how they are to be represented for purposes of conversion with the primitives in Section 5.5. As noted above, the internal representation of objects is left to the implementation, and may be different. If the internal representation is different, conversion to the representation defined here may be needed at certain points in cryptographic operations.

### 5.3.1 Prime Finite Fields

A *prime finite field* is a field containing a prime number of elements. If $p$ is an odd prime number, then there is a unique field $GF(p)$ with $p$ elements. For purposes of conversion, the elements of $GF(p)$ shall be represented by the integers 0, 1, 2..., $p - 1$.

A description of the arithmetic of $GF(p)$ is given in Annex A.1 and A.2.

### 5.3.2 Characteristic Two Finite Fields

A *characteristic two finite field* is a finite field whose number of elements is a power of 2. If $m \geq 1$, then there is a unique field $GF(2^m)$ with $2^m$ elements. For purposes of conversion, the elements of $GF(2^m)$ should be represented by bit strings of length $m$.

There are several ways of performing arithmetic in $GF(2^m)$. The specific rules depend on how the bit strings are interpreted. For purposes of conversion, this interpretation should be made in terms of one of the following *basis representations*.

NOTE—As opposed to the representation method for prime finite fields, which is required, the representation methods for binary finite fields are recommendations. The choice of a particular representation for binary fields affects the primitives and schemes in which conversion from field elements to other objects is used. See Annex D.4.1.3 and D.4.2.3 (and related notes in D.4.1.4 and D.4.2.4) for related security considerations.

#### 5.3.2.1 Polynomial Basis over *GF* (2)

This representation is determined by choosing an irreducible binary polynomial $p(t)$ of degree $m$. (See Annex A.3 and A.4 for the definitions of the above terms and for a description of the arithmetic of a field using this representation.) If the polynomial basis representation over $GF(2)$ is used, then, for purposes of conversion, the bit string

$$(a_{m-1} \ \ldots \ a_2 \ a_1 \ a_0)$$

shall be taken to represent the polynomial

$$a_{m-1}t^{m-1} + \ldots + a_2t^2 + a_1t + a_0$$

where the coefficients $a_i$ are elements of $GF(2)$.

In particular, for purposes of conversion, the additive identity (zero) element of the field is represented by a string of all zero bits, and the multiplicative identity (one) element of the field is represented by a string where all bits but the last are zero, and the last bit is one. (Note, however, that in mathematical expressions in this standard, for typographic convenience, the numeral 0 is used to represent the element zero of a field, and the numeral 1 is used to represent the element one of the field.)

NOTE—In keeping with the traditional mathematical notation, the bits in this representation are indexed from right to left, as opposed to the bits in the normal basis representation (Section 5.3.2.2), which are indexed from left to right. See also Section 5.2.

#### 5.3.2.2 Normal Basis over *GF* (2)

This representation is determined by choosing a normal polynomial $p(t)$ of degree $m$. (See Annex A.3 and A.4 for the definitions of the above terms and for a description of the arithmetic of a field using this

representation.) If the normal basis representation over *GF* (2) is used, then, for purposes of conversion, the bit string

$$(a_0\ a_1 \ldots\ a_{m-1})$$

shall be taken to represent the element

$$a_0 q + a_1 q^{2} + a_2 q^{2^2} + \ldots + a_{m-1} q^{2^{m-1}},$$

where $q$ is a root of $p(t)$ in $GF(2^m)$.

In particular, for purposes of conversion, the additive identity (zero) element of the field is represented by a string of all zero bits, and the multiplicative identity (one) element of the field is represented by a string of one bits. (Note, however, that in mathematical expressions in this standard, for typographic convenience, the numeral 0 is used to represent the element zero of a field, and the numeral 1 is used to represent the element one of the field.)

NOTE—In keeping with the traditional mathematical notation, the bits in this representation are indexed from left to right, as opposed to the bits in the polynomial basis representation (Section 5.3.2.1), which are indexed from right to left. See also Section 5.2.

## 5.4 Elliptic Curves and Points

An elliptic curve $E$ defined over $GF(q)$ is a set of points $P = (x_P, y_P)$ where $x_P$ and $y_P$ are elements of $GF(q)$ that satisfy a certain equation, together with the point at infinity denoted by $\mathsf{O}$. For the purposes of this standard, it is specified by two field elements $a \in GF(q)$ and $b \in GF(q)$, called the *coefficients* of $E$. The field elements $x_P$ and $y_P$ are called the $x$-coordinate of $P$ and the $y$-coordinate of $P$, respectively.

If $q$ is an odd prime, then $a$ and $b$ shall satisfy $4a^3 + 27b^2 \neq 0$ in $GF(q)$, and every point $P = (x_P, y_P)$ on $E$ (other than the point $\mathsf{O}$) shall satisfy the following equation in $GF(q)$:

$$y_P^2 = x_P^3 + a\,x_P + b.$$

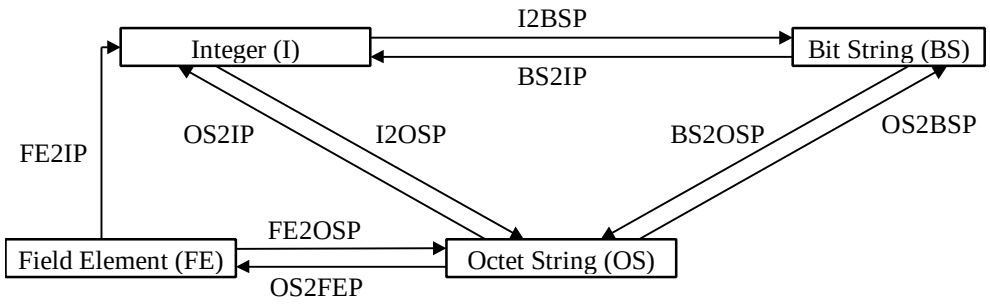
If $q$ is a power of 2, then $b$ shall be nonzero in $GF(q)$, and every point $P = (x_P, y_P)$ on $E$ (other than the point $\mathsf{O}$) shall satisfy the following equation in $GF(q)$:

$$y_P^2 + x_P\,y_P = x_P^3 + a\,x_P^2 + b.$$

See Annex A.9 and A.10 for more on elliptic curves and elliptic curve arithmetic.

## 5.5 Data Type Conversion

This section describes the primitives that shall be used to convert between different types of objects and strings when such conversion is required in primitives, schemes or encoding techniques. Representation of mathematical and cryptographic objects as octet strings is not specifically addressed here; rather, it is discussed in the informative Annex E. The following diagram describes the primitives presented in this section.




### 5.5.1 Converting Between Integers and Bit Strings (I2BSP and BS2IP)

In performing cryptographic operations, bit strings sometimes need to be converted to non-negative integers and vice versa.

To convert a non-negative integer $x$ to a bit string of length $l$ ($l$ has to be such that $2^l > x$), the integer shall be written in its unique $l$-digit representation base 2:

$$x = x_{l-1}\, 2^{l-1} + x_{l-2}\, 2^{l-2} + \ldots + x_1\, 2 + x_0$$

where $x_i$ is either 0 or 1 (note that one or more leading digits will be zero if $x < 2^{l-1}$). Then let the bit $b_i$ have the value $x_{l-i}$ for $1 \le i \le l$. The bit string shall be $b_1\, b_2 \ldots b_l$.

For example, the integer 10945 is represented by a bit string of length 19 as 000 0010 1010 1100 0001.

The primitive that converts integers to bit strings is called Integer to Bit String Conversion Primitive or I2BSP. It takes an integer $x$ and the desired length $l$ as input and outputs the bit string if $2^l > x$. It shall output "error" otherwise.

The primitive that converts bit strings to integers is called Bit String to Integer Conversion Primitive or BS2IP. It takes a bit string as input and outputs the corresponding integer. Note that the bit string of length zero (the empty bit string) is converted to the integer 0.

### 5.5.2 Converting Between Bit Strings and Octet Strings (BS2OSP and OS2BSP)

To represent a bit string as an octet string, one simply pads enough zeroes on the left to make the number of bits a multiple of 8, and then breaks it up into octets. More precisely, a bit string $b_{l-1}\, b_{l-2} \ldots b_0$ of length $l$ shall be converted to an octet string $M_{d-1}\, M_{d-2} \ldots M_0$ of length $d = \lceil l/8 \rceil$ as follows: for $0 \le i < d - 1$, let the octet $M_i = b_{8i+7}\, b_{8i+6} \ldots b_{8i}$. The leftmost octet $M_{d-1}$ shall have its leftmost $8d - l$ bits set to 0; its rightmost $8 - (8d - l)$ bits shall be $b_{l-1}\, b_{l-2} \ldots b_{8d-8}$.

The primitive that converts bit strings to octet strings is called Bit String to Octet String Conversion Primitive or BS2OSP. It takes the bit string as input and outputs the octet string.

The primitive that converts octet strings to bit strings is called Octet String to Bit String Conversion Primitive or OS2BSP. It takes an octet string of length $d$ and the desired length $l$ of the bit string as input. It shall output the bit string if $d = \lceil l/8 \rceil$ and if the leftmost $8d - l$ bits of the leftmost octet are zero; it shall output "error" otherwise.

### 5.5.3 Converting between Integers and Octet Strings (I2OSP and OS2IP)

To represent a non-negative integer $x$ as an octet string of length $l$ ($l$ has to be such that $256^l > x$), the integer shall be written in its unique $l$-digit representation base 256:

$$x = x_{l-1}\, 256^{l-1} + x_{l-2}\, 256^{l-2} + \ldots + x_1\, 256 + x_0$$

where $0 \le x_i < 256$ (note that one or more leading digits will be zero if $x < 256^{l-1}$). Then let the octet $M_i$ have the value $x_i$ for$0 \le i \le l$-1. The octet string shall be $M_{l-1}\, M_{l-2} \ldots M_0$.

For example, the integer 10945 is represented by an octet string of length 3 as 00 2A C1.

The primitive that converts integers to octet strings is called Integer to Octet String Conversion Primitive or I2OSP. It takes an integer $x$ and the desired length $l$ as input and outputs the octet string if $256^l > x$. It shall output "error" otherwise.

The primitive that converts octet strings to integers is called Octet String to Integer Conversion Primitive or OS2IP. It takes an octet string as input and outputs the corresponding integer. Note that the octet string of length zero (the empty octet string) is converted to the integer 0.

### 5.5.4 Converting between Finite Field Elements and Octet Strings (FE2OSP and OS2FEP)

An element $x$ of a finite field $GF(q)$, for the purposes of this standard, is represented by an integer if $q$ is an odd prime (see Section 5.3.1) or by a bit string if $q$ is a power of 2 (see Section 5.3.2). If $q$ is an odd prime, then to represent $x$ as an octet string, I2OSP shall be used with the integer value of $x$ and the length $\lceil \log_{256} q \rceil$ as inputs. If $q$ is a power of 2, then to represent $x$ as an octet string, BS2OSP shall be applied to the bit string representing $x$.

The primitive that converts finite field elements to octet strings is called Field Element to Octet String Conversion Primitive or FE2OSP. It takes a field element $x$ and the field size $q$ as inputs and outputs the corresponding octet string.

To convert an octet string back to a field element, if $q$ is an odd prime, then OS2IP shall be used with the octet string as the input. If $q$ is a power of 2, then OS2BSP shall be used with the octet string and the length $\log_2 q$ as inputs. The primitive that converts octet strings to finite field elements is called Octet String to Field Element Conversion Primitive or OS2FEP. It takes the octet string and the field size $q$ as inputs and outputs the corresponding field element. It shall output "error" if OS2BSP or OS2IP outputs "error."

### 5.5.5 Converting Finite Field Elements to Integers (FE2IP)

In performing cryptographic operations, finite field elements sometimes need to be converted to non-negative integers. The primitive that performs this is called Field Element to Integer Conversion Primitive or FE2IP.

A element $j$ of a finite field $GF(q)$ shall be converted to a non-negative integer $i$ by the following, or an equivalent, procedure:

1. Convert the element $j$ to an octet string using FE2OSP.
2. Convert the resulting octet string to an integer $i$ using OS2IP.
3. Output $i$.

Note that if $q$ is an odd prime, $j$ is already represented as an integer and FE2IP merely outputs that representation. If $q$ is a power of 2, then FE2IP outputs an integer whose binary representation is the same as the bit string representing $j$.

# 6. Primitives Based on the Discrete Logarithm Problem

This section specifies the family of cryptographic primitives based on the discrete logarithm problem over finite fields, also known as the DL family. For background information on this family, see [DH76], Annex A.1.2 and A.3.9.

## 6.1 The DL Setting

### 6.1.1 Notation

The list below introduces the notation used in Sections 6, 9, and 10. It is meant as a reference guide only; for complete definitions of the terms listed, refer to the appropriate text. Some other symbols are also used occasionally; they are introduced in the text where appropriate.

| | |
|---|---|
| $q$ | The size of the field used (part of the DL domain parameters) |
| $r$ | The prime divisor of $q - 1$ and the order of $g$ (part of the DL domain parameters) |
| $g$ | An element of the field $GF(q)$ generating a multiplicative subgroup of order $r$ (part of the DL domain parameters) |
| $k$ | $(q - 1) / r$, the cofactor |
| $s$, $u$, $s'$, $u'$ | DL private keys, integers, corresponding to public keys $w$, $v$, $w'$, $v'$ respectively |
| $w$, $v$, $w'$, $v'$ | DL public keys, elements of $GF(q)$, corresponding to private keys $s$, $u$, $s'$, $u'$ respectively |
| $(s, w)$, $(u, v)$ | DL key pairs, where $s$ and $u$ are private keys, and $w$ and $v$ are the corresponding public keys |
| $z$, $z_1$, $z_2$ | Shared secret values, elements of $GF(q)$, derived by secret value derivation primitives |
| $K$ | Shared secret key agreed upon by a key agreement scheme |
| $f$ | Message representative, an integer, computed by a message encoding operation |
| $M$ | The message, an octet string, whose signature is computed by a signature scheme |

NOTES

1—When keys from two parties are involved in a primitive or scheme, the symbols $s$, $u$, $w$, $v$ are used to denote the party's own keys, and the symbols $s'$, $u'$, $w'$, $v'$ are used to denote the other party's keys.

2—Multiplication of field elements, as well as multiplication of integers, is denoted by $\times$, although this symbol may be omitted when such omission does not cause ambiguity. The notation such as $\exp(w, c)$, where $w$ is a field element and $c$ is an integer, will be used to denote raising $w$ to the power $c$. The more traditional notation $w^c$ will be used if $w$ is an integer.

3—Throughout this section, operations on finite field elements and integers are being used. Care needs to be exercised to distinguish integers from field elements, as operations on integers and field elements are denoted by the same symbols. See Section 5.3 for more information on finite fields.

### 6.1.2 DL Domain Parameters

*DL domain parameters* are used in every DL primitive and scheme and are an implicit component of every DL key. A set of DL domain parameters specifies a field $GF(q)$, where $q$ is a positive odd prime integer $p$ or $2^m$ for some positive integer $m$; a positive prime integer $r$ dividing $q - 1$; and a field element $g$ of multiplicative order $r$ ($g$ is called the *generator* of the subgroup of order $r$). If $q = 2^m$, it also specifies a representation for the elements of $GF(q)$ to be used by the conversion primitives (see Section 5.3 and 5.5). Implicitly it also specifies the cofactor $k = (q - 1) / r$. If the DLSVDP-DHC or DLSVDP-MQVC primitive

is to be applied, then it shall also be the case that GCD ($k$, $r$) = 1 (i.e., $r$ does not divide $k$; see Annex A.1.1).

Depending on the scheme and protocol used, a party may need to generate its own set of domain parameters or use domain parameters provided by another party. If parameters are provided by another party, their authenticity may need to be determined (as further discussed in Annex D.3.1), and they may need to be validated (see below). The security issues related to domain parameter generation are discussed in D.3.2 and D.4.1. Suggested methods for generating DL domain parameters are contained in Annex A.16.1 and A.16.3.

Parties establish DL domain parameters as part of key management for a scheme. Depending on the key management technique, it is possible that the established domain parameters do not satisfy the intended definition, even though they have the same general form (i.e., components $q$, $r$, $g$ and optionally $k$). To accommodate this possibility, the term "DL domain parameters" shall be understood in this standard as referring to instances of the general form for DL domain parameters. The term *valid DL domain parameters* shall be reserved for DL domain parameters satisfying the definition.

*Domain parameter validation* is the process of determining whether a set of DL domain parameters is valid. Further discussion of domain parameter validation is contained in D.3.3. Suggested algorithms for DL domain parameter validation are contained in Annex A.16.2 and A.16.4.

There may be more than one set of domain parameters used in a primitive or scheme; the sets of DL domain parameters may be different for different keys, or they may be the same, depending on the requirements of a primitive or scheme. Unlike keys, which are not meant to be shared among users, a set of domain parameters can and sometimes needs to be shared. DL domain parameters are often public; the security of the schemes in this standard does not rely on these domain parameters being secret.

### 6.1.3 DL Key Pairs

For a given set of DL domain parameters, a *DL key pair* consists of a *DL private key* $s$ which is an integer in the range $[1, r - 1]$ and a *DL public key* $w$ which is an element of $GF(q)$, where $w = \exp(g, s)$. For security, DL private keys need to be generated so that they are unpredictable and stored so that they are inaccessible to an adversary. DL private keys may be stored in any form convenient to the application.

DL key pairs are closely associated with their domain parameters, and can only be used in the context of the domain parameters. A key pair shall not be used with a set of domain parameters different from the one for which it was generated. A set of domain parameters may be shared by a number of key pairs (see Annex D.4.1.2 and D.4.1.4, Note 1).

A DL key pair may or may not be generated by the party using it, depending on the trust model. This and other security issues related to DL key pair generation are discussed in D.3.2 and D.4.1. A suggested method for generating DL key pairs is contained in A.16.5.

As is also the case for DL domain parameters, parties establish DL keys as part of key management for a scheme, and depending on the key management technique, it is possible that an established key does not satisfy the intended definition for the key, even though it has the same general form. Accordingly, the terms "DL public key" and "DL private key" shall be understood in this standard as referring to instances of the general form for the key, and the terms *valid DL public key*, *valid DL private key*, and *valid DL key pair* shall be reserved to keys satisfying the definition.

*Key validation* is the process of determining whether a key is valid. Further discussion of key validation is contained in D.3.3. A suggested algorithm for DL public key validation is contained in Annex A.16.6. In some cases (when DLSVDP-DHC or DLSVDP-MQVC primitive is applied), it may only be necessary

to verify that the public key is a member of $GF(q)$ (rather than a stronger condition that it is a power of $g$ other than one). The algorithm to verify this weaker condition is also contained in Annex A.16.6. No algorithm is given for DL private key validation, because, generally, a party controls its own private key and need not validate it. However, private key validation may be performed if desired.

## 6.2 Primitives

This section describes primitives used in the DL family. Before proceeding with this section, the reader should be familiar with the material of Section 4.

As detailed in Section 4 and Annex B, an implementation of a primitive may make certain assumptions about its inputs, as listed with the specification of the primitive. For example, if DL domain parameters $q$, $r$ and $g$ and a public key $w$ are inputs to a primitive, an implementation may generally assume that the domain and the public key parameters are valid, i.e., that $g$ has order $r$ in $GF(q)$ and $w = \exp(g, s)$ for some integer $s$ in the interval $[1, r - 1]$; the behavior of an implementation is not specified if $w$ is not a power of $g$, and in such a case the implementation may or may not output an error condition. It is up to the properly implemented scheme to ensure that only appropriate inputs are passed to a primitive, or to accept the risks of passing inappropriate inputs. For more on conforming with a primitive, see Annex B.

### 6.2.1 DLSVDP-DH

DLSVDP-DH is Discrete Logarithm Secret Value Derivation Primitive, Diffie-Hellman version. It is based on the work of [DH76]. This primitive derives a shared secret value from one party's public key and another party's private key, where the keys have the same set of DL domain parameters. If two parties correctly execute this primitive with corresponding keys as inputs, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the schemes DLKAS-DH1 and DL/ECKAS-DH2. It assumes that the input keys are valid (see also Section 6.2.2).

**Input:**
- the DL domain parameters $q$, $r$ and $g$ associated with the keys $s$ and $w$ (the domain parameters shall be the same for both $s$ and $w$)
- the party's own private key $s$
- the other party's public key $w$

**Assumptions:** private key $s$, DL domain parameters $q$, $r$ and $g$, and public key $w$ are valid; both keys are associated with the domain parameters

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF(q)$

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. Compute a field element $z = \exp(w, s)$.
2. Output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
- at least one valid set of DL domain parameters $q$, $r$ and $g$
- at least one valid private key $s$ for each set of domain parameters
- all valid public keys $w$ associated with the same set of domain parameters as $s$

NOTES

1—This primitive does not address small subgroup attacks (see Annex D.5.1.6), which may occur when the public key $w$ is not valid. To prevent them, a key agreement scheme should validate the public key $w$ before executing this primitive (see also Section 6.2.2).

2—When the public key $w$ and the private key $s$ are valid, $w$ has order $r$ and $s$ is in the interval [1, $r$-1]. Thus the output $z$ in this case is neither the element zero nor the element one.

### 6.2.2 DLSVDP-DHC

DLSVDP-DHC is Discrete Logarithm Secret Value Derivation Primitive, Diffie-Hellman version with cofactor exponentiation. It is based on the work of [DH76], [LMQ98] and [Kal98a]. This primitive derives a shared secret value from one party's public key and another party's private key, where the keys have the same set of DL domain parameters. If two parties correctly execute this primitive with corresponding keys as inputs, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the schemes DLKAS-DH1 and DL/ECKAS-DH2. It does not assume the validity of the input public key, but does assume that the public key is an element of GF($q$) (see also Section 6.2.1).

**Input:**
— the DL domain parameters $q$, $r$, $g$ and the cofactor $k$ associated with the keys $s$ and $w$ (the domain parameters shall be the same for both $s$ and $w$);
— the party's own private key $s$
— the other party's public key $w$
— an indication as to whether compatibility with DLSVDP-DH is desired

**Assumptions:** private key $s$ and DL domain parameters $q$, $r$, $g$ and $k$ are valid; the private key is associated with the domain parameters; $w$ is in GF ($q$); GCD ($k$, $r$) = 1

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF(q)$; or "invalid public key"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. If compatibility with DLSVDP-DH is desired, then compute an integer $t = k^{-1}s \bmod r$; otherwise set $t = s$.
2. Compute a field element $z = \exp(w, kt)$.
3. If $z = 0$ or $z = 1$, output "invalid public key"; else, output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of DL domain parameters $q$, $r$, $g$ and $k$
— at least one valid private key $s$ for each set of domain parameters
— all elements $w$ in GF ($q$), where $q$ is from the domain parameters of $s$
— compatibility with DLSVDP-DH may be preset by the implementation, or given as an input flag

NOTES

1—This primitive addresses small subgroup attacks, which may occur when the public key $w$ is not valid (see Annex D.5.1.6). A key agreement scheme only needs to validate that $w$ is in GF ($q$) before executing this primitive (see also Section 6.2.1).

2—The cofactor $k$ depends only on the DL domain parameters and is equal to $(q - 1) / r$. Hence, it can be computed once for a given set of domain parameters and stored as part of the domain parameters. Similarly, in the compatibility

case, the value $k^{-1}$ can be computed once and stored with the domain parameters, and the integer $t$ can be computed once for a given private key $s$.

3—When the public key $w$ and the private key $s$ are valid, the output $z$ will be an element of a subset of $GF(q)$ that consists of all the powers of $g$ (except for the element one). As a consequence, $z$ cannot be the element zero or the element one in this case. When the public key is invalid, the output will be either "invalid public key" or an element of order r in $GF(q)$; in particular, it will not be in a small subgroup.

4—In the compatibility case, DLSVDP-DHC computes the same output for valid keys as DLSVDP-DH, so an implementation that conforms with DLSVDP-DHC in the compatibility case also conforms with DLSVDP-DH.

### 6.2.3 DLSVDP-MQV

DLSVDP-MQV is Discrete Logarithm Secret Value Derivation Primitive, Menezes-Qu-Vanstone version. It is based on the work of [LMQ98]. This primitive derives a shared secret value from one party's two key pairs and another party's two public keys, where all the keys have the same set of DL domain parameters. If two parties execute this primitive with corresponding keys as inputs, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the scheme DLKAS-MQV. It assumes that the input keys are valid (see also Section 6.2.4).

In this primitive, let $h = \lceil (\log_2 r) / 2 \rceil$. Note that $h$ depends only on the DL domain parameters, and hence can be computed once for a given set of domain parameters.

**Input:**
- the DL domain parameters $q$, $r$ and $g$ associated with the keys $s$, $(u, v)$, $w$ and $v$ (the domain parameters shall be the same for these keys)
- the party's own first private key $s$
- the party's own second key pair $(u, v)$
- the other party's public key $w$
- the other party's second public key $v$

**Assumptions:** private key $s$, key pair $(u, v)$, public keys $w$ $v$ and DL domain parameters $q$, $r$ and $g$ are valid; all the keys are associated with the domain parameters

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF(q)$; or "error"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. Convert $v$ into an integer $i$ using FE2IP.
2. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.
3. Convert $v$ into an integer $i$ using FE2IP.
4. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.
5. Compute an integer $e = ts + u \bmod r$.
6. Compute a field element $z = \exp(v \cdot \exp(w, t), e)$.
7. If $z = 1$, output "error" and stop.
8. Output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
- at least one valid set of DL domain parameters $q$, $r$ and $g$
- at least one valid private key $s$ for each set of domain parameters
- all valid key pairs $(u, v)$ associated with the same set of domain parameters as $s$
- all valid public keys $w$ and $v$ associated with the same set of domain parameters as $s$

NOTES

1—This primitive does not address small subgroup attacks (see Annex D.5.1.6), which may occur when the public keys $w_0$ and $v_0$ are not valid. To prevent them, a key agreement scheme should validate the public keys $w_0$ and $v_0$ before executing this primitive (see also Section 6.2.4).

2—When the public keys $w_0$ and $v_0$ are valid, the output $z$ will be an element of a subset of $GF(q)$ that consists of all the powers of $g$. It is possible (although very unlikely) that even though all the keys and parameters are valid, $z$ will be the element one. In this case, the primitive will output "error," and will need to be re-run with a different input. Except for this rare case, $z$ will always be defined and will not be the element zero or the element one as long as the input keys and parameters are valid.

### 6.2.4 DLSVDP-MQVC

DLSVDP-MQVC is Discrete Logarithm Secret Value Derivation Primitive, Menezes-Qu-Vanstone version with cofactor exponentiation. It is based on the work of [LMQ98] and [Kal98a]. This primitive derives a shared secret value from one party's two key pairs and another party's two public keys, where all the keys have the same set of DL domain parameters. If two parties execute this primitive with corresponding keys as inputs, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the scheme DLKAS-MQV. It does not assume the validity of the input public keys, but does assume that the public key is an element of GF($q$) (see also Section 6.2.3).

In this primitive, let $h = \lceil (\log_2 r) / 2 \rceil$. Note that $h$ depends only on the DL domain parameters, and hence can be computed once for a given set of domain parameters.

**Input:**

— the DL domain parameters $q$, $r$, $g$ and the cofactor $k$ associated with the keys $s$, $(u, v)$, $w_0$ and $v_0$ (the domain parameters shall be the same for these keys)
— the party's own first private key $s$
— the party's own second key pair $(u, v)$
— the other party's public key $w_0$
— the other party's second public key $v_0$
— an indication as to whether compatibility with DLSVDP-MQV is desired

**Assumptions:** private key $s$, key pair $(u, v)$, and DL domain parameters $q$, $r$, $g$ and $k$ are valid; private key $s$ and key pair $(u, v)$ are associated with the domain parameters; $w_0$ and $v_0$ are in GF ($q$); GCD ($k$, $r$) = 1

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF(q)$; or "invalid public key"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. Convert $v$ into an integer $i$ using FE2IP.
2. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.
3. Convert $v_0$ into an integer $i_0$ using FE2IP.
4. Compute an integer $t_0 = i_0 \bmod 2^h$. Let $t_0 = t_0 + 2^h$.
5. If compatibility with DLSVDP-MQV is desired, then compute an integer $e = k^{-1}(ts + u) \bmod r$; otherwise compute an integer $e = ts + u \bmod r$.
6. Compute a field element $z = \exp(v_0 \times \exp(w_0, t_0), ke)$.
7. If $z = 0$ or $z = 1$, output "invalid public key"; else, output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of DL domain parameters $q$, $r$, $g$ and $k$
— at least one valid private key $s$ for each set of domain parameters
— all valid key pairs ($u$, $v$) associated with the same set of domain parameters as $s$
— all elements $w$ ¢and $v$ ¢in GF ($q$), where $q$ is from the domain parameters of $s$
— compatibility with DLSVDP-MQV may be preset by the implementation, or given as an input flag

NOTES

1—This primitive addresses small subgroup attacks, which may occur when the public keys $w$ ¢and $v$ ¢are not valid (see Annex D.5.1.6). A key agreement scheme only needs to validate that $w$ ¢and $v$ ¢are in GF ($q$) before executing this primitive (see also Section 6.2.3).

2—The cofactor $k$ depends only on the DL domain parameters and is equal to $(q - 1) / r$. Hence, it can be computed once for a given set of domain parameters and stored as part of the domain parameters. Similarly, in the compatibility case, the value $k^{-1}$ can be computed once and stored with the domain parameters. An equivalent way to compute the integer $e$ in this case is as $(tk^{-1}s + k^{-1}u) \bmod r$, where $k^{-1}s \bmod r$ and $k^{-1}u \bmod r$ can be computed once for each given private key $s$ and $u$.

3—When the public keys $w$ ¢and $v$ ¢are valid, the output $z$ will be an element of a subset of GF ($q$) that consists of all the powers of $g$. It is possible (although very unlikely) that even though all the keys and parameters are valid, $z$ will be the element one, and the primitive will output "invalid public key." In this case, the primitive will need to be re-run with a different input. Except for this rare case, $z$ will always be defined and will not be the element zero or the element one as long as the input keys and parameters are valid. When one of the public keys is invalid, the output will be either "invalid public key" or an element of order $r$ in GF ($q$); in particular, it will not be in a small subgroup.

4—In the compatibility case, DLSVDP-MQVC computes the same output for valid keys as DLSVDP-MQV, so an implementation that conforms with DLSVDP-MQVC in the compatibility case also conforms with DLSVDP-MQV.

### 6.2.5 DLSP-NR

DLSP-NR is Discrete Logarithm Signature Primitive, Nyberg-Rueppel version. It is based on the work of [NR93]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the DLVP-NR primitive. Note, however, that no DL signature schemes with message recovery are defined in this version of the standard (see Annex C.3.4). DLSP-NR may also be used in a signature scheme with appendix and can be invoked in the scheme DLSSA as part of signature generation.

**Input:**
— the DL domain parameters $q$, $r$ and $g$ associated with the key $s$
— the signer's private key $s$
— the message representative, which is an integer $f$ such that $0 \le f < r$

**Assumptions:** private key $s$ and DL domain parameters $q$, $r$ and $g$ are valid and associated with each other; $0 \le f < r$

**Output:** the signature, which is a pair of integers ($c$, $d$), where $1 \le c < r$ and $0 \le d < r$

**Operation.** The signature ($c$, $d$) shall be computed by the following or an equivalent sequence of steps:

1. Generate a key pair ($u$, $v$) with the same set of domain parameters as the private key $s$. (See the note below.)
2. Convert $v$ into an integer $i$ with primitive FE2IP (recall that $v$ is an element of $GF$ ($q$)).

3. Compute an integer $c = i + f \bmod r$. If $c = 0$, then go to Step 1.
4. Compute an integer $d = u - sc \bmod r$.
5. Output the pair $(c, d)$ as the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of DL domain parameters $q$, $r$ and $g$
— at least one valid private key $s$ for each set of domain parameters
— all message representatives $f$ in the range $[0, r - 1]$ , where $r$ is from the domain parameters of $s$

NOTE—The key pair in Step 1 should be a one-time key pair which is generated and stored by the signer following the security recommendations of D.3.2, D.4.1.2, D.6 and D.7. A new key pair should be generated for every signature. The one-time private key $u$ should be discarded after step 4, as its recovery by an opponent can lead to the recovery of the private key $s$.

### 6.2.6 DLVP-NR

DLVP-NR is Discrete Logarithm Verification Primitive, Nyberg-Rueppel version. It is based on the work of [NR93]. This primitive recovers the message representative that was signed with DLSP-NR, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that no DL signature schemes with message recovery are defined in this version of the standard (see Annex C.3.4). DLVP-NR may also be used in a signature scheme with appendix and can be invoked in the scheme DLSSA as part of signature verification.

**Input:**
— the DL domain parameters $q$, $r$ and $g$ associated with the key $w$
— the signer's public key $w$
— the signature to be verified, which is a pair of integers $(c, d)$

**Assumptions:** public key $w$ and DL domain parameters $q$, $r$ and $g$ are valid and associated with each other

**Output:** the message representative, which is an integer $f$ such that $0 \le f < r$; or "invalid"

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1. If $c$ is not in $[1, r - 1]$ or $d$ is not in $[0, r - 1]$, output "invalid" and stop.
2. Compute a field element $j = \exp(g, d) \times \exp(w, c)$.
3. Convert the field element $j$ to an integer $i$ with primitive FE2IP.
4. Compute an integer $f = c - i \bmod r$.
5. Output $f$ as the message representative.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of DL domain parameters $q$, $r$ and $g$
— at least one valid public key $w$ for each set of domain parameters
— all purported signatures $(c, d)$ that can be input to the implementation; this should at least include all $(c, d)$ such that $c$ and $d$ are in the range $[0, r - 1]$ , where $r$ is from the domain parameters of $w$

### 6.2.7 DLSP-DSA

DLSP-DSA is Discrete Logarithm Signature Primitive, DSA version. It is based on the work of [Kra93]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer. The message representative cannot be recovered from the signature, but DLVP-DSA can be used in the scheme DLSSA to verify the signature.

**Input:**
— the DL domain parameters $q$, $r$ and $g$ associated with the key $s$
— the signer's private key $s$
— the message representative, which is an integer $f \geq 0$

**Assumptions:** private key $s$ and DL domain parameters $q$, $r$ and $g$ are valid and associated with each other; $f \geq 0$

**Output:** the signature, which is a pair of integers $(c, d)$, where $1 \leq c < r$ and $1 \leq d < r$

**Operation.** The signature $(c, d)$ shall be computed by the following or an equivalent sequence of steps:

1. Generate a key pair $(u, v)$ with the same set of domain parameters as the private key $s$. (See the note below.)
2. Convert $v$ into an integer $i$ with primitive FE2IP (recall that $v$ is an element of $GF(q)$).
3. Compute an integer $c = i \bmod r$. If $c = 0$, then go to Step 1.
4. Compute an integer $d = u^{-1}(f + sc) \bmod r$. If $d = 0$, then go to Step 1.
5. Output the pair $(c, d)$ as the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of DL domain parameters $q$, $r$ and $g$
— at least one valid private key $s$ for each set of domain parameters
— a range of message representatives $f$; this should at least include all $f \geq 0$ with bit length no greater than that of $r$, where $r$ is from the domain parameters of $s$

NOTE—The key pair in Step 1 should be a one-time key pair which is generated and stored by the signer following the security recommendations of D.3.2, D.4.1.2, D.6 and D.7. A new key pair should be generated for every signature. The one-time private key $u$ should be discarded after step 4, as its recovery by an opponent can lead to the recovery of the private key $s$.

### 6.2.8 DLVP-DSA

DLVP-DSA is Discrete Logarithm Verification Primitive, DSA version. It is based on the work of [Kra93]. This primitive verifies whether the message representative and the signature are consistent given the key and the domain parameters. It can be invoked in the scheme DLSSA as part of signature verification.

**Input:**
— the DL domain parameters $q$, $r$ and $g$ associated with the key $w$
— the signer's public key $w$
— the message representative, which is an integer $f \geq 0$
— the signature to be verified, which is a pair of integers $(c, d)$

**Assumptions:** public key $w$ and DL domain parameters $q$, $r$ and $g$ are valid and associated with each other; $f \geq 0$

**Output:** "valid" if $f$ and $(c, d)$ are consistent given the key and the domain parameters; "invalid" otherwise

**Operation.** The output shall be computed by the following or an equivalent sequence of steps:

1. If $c$ is not in $[1, r - 1]$ or $d$ is not in $[1, r - 1]$, output "invalid" and stop.
2. Compute integers $h = d^{-1} \bmod r$; $h_1 = f\,h \bmod r$; $h_2 = c\,h \bmod r$.

3. Compute a field element $j = \exp(g, h_1) \times \exp(w, h_2)$.
4. Convert the field element $j$ to an integer $i$ with primitive FE2IP.
5. Compute an integer $c \Leftarrow i \bmod r$.
6. If $c \Leftarrow c$, then output "valid"; else, output "invalid."

**Conformance region recommendation.** A conformance region should include:

— at least one valid set of DL domain parameters $q$, $r$ and $g$
— at least one valid public key $w$ for each set of domain parameters
— all message representatives $f \geq 0$ that can be input to the implementation; this should at least include all $f$ with bit length no greater than that of $r$, where $r$ is from the domain parameters of $w$
— all purported signatures $(c, d)$ that can be input to the implementation; this should at least include all $(c, d)$ such that $c$ and $d$ are in the range $[1, r - 1]$ , where $r$ is from the domain parameters of $w$

# 7. Primitives Based on the Elliptic Curve Discrete Logarithm Problem

This section specifies the family of cryptographic primitives based on the discrete logarithm problem over elliptic curve groups, also known as the EC family. For background information on this family, see [Mil86], [Kob87], Annex A.9.3 and A.9.4.

## 7.1 The EC Setting

### 7.1.1 Notation

The list below introduces the notation used in Sections 7, 9, and 10. It is meant as a reference guide only; for complete definitions of the terms listed, refer to the appropriate text. Some other symbols are also used occasionally; they are introduced in the text where appropriate.

| | |
|---|---|
| $q$ | The size of the underlying field used (part of the EC domain parameters) |
| $a$, $b$ | The coefficients defining the elliptic curve $E$, elements of $GF(q)$ (part of the EC domain parameters) |
| $E$ | The elliptic curve over the field $GF(q)$ defined by $a$ and $b$ |
| $\#E$ | The number of points on the elliptic curve $E$ |
| $r$ | The prime divisor of $\#E$ and the order of $G$ (part of the EC domain parameters) |
| $G$ | A curve point generating a subgroup of order $r$ (part of the EC domain parameters) |
| $k$ | #E / $r$, the cofactor |
| $s$, $u$, $s'$, $u'$ | EC private keys, integers, corresponding to public keys $W$, $V$, $W'$, $V'$ respectively |
| $W$, $V$, $W'$, $V'$ | EC public keys, points on the curve, corresponding to private keys $s$, $u$, $s'$, $u'$ respectively |
| $(s, W)$, $(u, V)$ | EC key pairs, where $s$ and $u$ are the private keys, and $W$ and $V$ are the corresponding public keys |
| $z$, $z_1$, $z_2$ | Shared secret values, elements of $GF(q)$, derived by secret value derivation primitives |
| $K$ | Shared secret key agreed upon by a key agreement scheme |
| $f$ | Message representative, an integer, computed by a message encoding operation |
| $M$ | The message, an octet string, whose signature is computed by a signature scheme |

NOTES

1—When keys from two parties are involved in a primitive or scheme, the symbols $s$, $u$, $W$, $V$ are used to denote the party's own keys, and the symbols $s'$, $u'$, $W'$, $V'$ are used to denote the other party's keys.

2—Multiplication of field elements, multiplication of integers, and scalar multiplication of elliptic curve points by integers are denoted by $\times$, although this symbol may be omitted when such omission does not cause ambiguity. Addition of field elements, addition of integers, and addition of elliptic curve points are denoted by +. Elliptic curve points are always denoted by capital letters; for an elliptic curve point $P \neq 0$, its $x$-coordinate and $y$-coordinate are denoted by $x_P$ and $y_P$, respectively: $P = (x_P, y_P)$.

3—Throughout this section, operations on finite field elements, integers, and elliptic curve points are used. Care needs to be exercised to distinguish these, as operations are denoted by the same symbols regardless of operands. See Section 5.3 for more information on finite fields; Section 5.4 for more information on elliptic curves.

### 7.1.2 EC Domain Parameters

*EC domain parameters* are used in every EC primitive and scheme and are an implicit component of every EC key. A set of EC domain parameters specifies a field $GF(q)$, where $q$ is a positive odd prime integer $p$ or $2^m$ for some positive integer $m$; two elliptic curve coefficients $a$ and $b$, elements of $GF(q)$, that

define an elliptic curve $E$; a positive prime integer $r$ dividing the number of points on $E$; and a curve point $G$ of order $r$ ($G$ is called the *generator* of a subgroup of order $r$). If $q = 2^m$, it also specifies a representation for the elements of $GF(q)$ to be used by the conversion primitives (see Section 5.3 and 5.5). Implicitly it also specifies the cofactor $k = \#E / r$ (where $\#E$ is the number of points on $E$). If key validation is to be performed or if the ECSVDP-DHC or ECSVDP-MQVC primitive is to be applied, then it shall also be the case that GCD ($k$, $r$) = 1 (i.e., $r$ does not divide $k$; see Annex A.1.1).

Depending on the scheme and protocol used, a party may need to generate its own set of domain parameters or use domain parameters provided by another party. If parameters are provided by another party, their authenticity may need to be determined (as further discussed in Annex D.3.1), and they may need to be validated (see below). The security issues related to domain parameter generation are discussed in D.3.2 and D.4.2. A suggested method for generating EC domain parameters is contained in Annex A.16.7 (see also Annex A.9.5).

Parties establish EC domain parameters as part of key management for a scheme. Depending on the key management technique, it is possible that the established domain parameters do not satisfy the intended definition, even though they have the same general form (i.e., components $q$, $r$, $G$ and optionally $k$). To accommodate this possibility, the term "EC domain parameters" shall be understood in this standard as referring to instances of the general form for EC domain parameters. The term *valid EC domain parameters* shall be reserved for EC domain parameters satisfying the definition.

*Domain parameter validation* is the process of determining whether a set of EC domain parameters is valid. Further discussion of domain parameter validation is contained in D.3.3. A suggested algorithm for EC domain parameter validation is contained in Annex A.16.8.

There may be more than one set of domain parameters used in a primitive or scheme; the sets of EC domain parameters may be different for different keys, or they may be the same, depending on the requirements of a primitive or scheme. Unlike keys, which are not meant to be shared among users, a set of domain parameters can and sometimes needs to be shared. EC domain parameters are often public; the security of the schemes in this standard does not rely on these domain parameters being secret.

### 7.1.3 EC Key Pairs

For a given set of EC domain parameters, an *EC key pair* consists of an *EC private key* $s$ which is an integer in the range $[1, r-1]$ and an *EC public key* $W$ which is a point on $E$, where $W = s\,G$ (note that $W \neq 0$, since $G$ has order $r$ and $0 < s < r$). For security, EC private keys need to be generated so that they are unpredictable and stored so that they are inaccessible to an adversary. EC private keys may be stored in any form convenient to the application.

EC key pairs are closely associated with their domain parameters, and can only be used in the context of the domain parameters. A key pair shall not be used with a set of domain parameters different from the one for which it was generated. A set of domain parameters may be shared by a number of key pairs (see Annex D.4.2.2 and D.4.2.4, Note 1).

An EC key pair may or may not be generated by the party using it, depending on the trust model. This and other security issues related to EC key pair generation are discussed in D.3.2 and D.4.2. A suggested method for generating EC key pairs is contained in Annex A.16.9.

As is also the case for EC domain parameters, parties establish EC keys as part of key management for a scheme, and depending on the key management technique, it is possible that an established key does not satisfy the intended definition for the key, even though it has the same general form. Accordingly, the terms "EC public key" and "EC private key" shall be understood in this standard as referring to instances

of the general form for the key, and the terms *valid EC public key*, *valid EC private key* and *valid EC key pair* shall be reserved to keys satisfying the definition.

*Key validation* is the process of determining whether a key is valid. Further discussion of key validation is contained in D.3.3. A suggested algorithm for EC public key validation is contained in Annex A.16.10. In some cases (when ECSVDP-DHC or ECSVDP-MQVC primitive is applied), it may only be necessary to verify that the public key is a point on the curve (rather than a stronger condition that it is a non-zero multiple of $G$). The algorithm to verify that is also contained in Annex A.16.10. No algorithm is given for EC private key validation, because, generally, a party controls its own private key and need not validate it. However, private key validation may be performed if desired.

## 7.2 Primitives

This section describes primitives used in the EC family. Before proceeding with this section, the reader should be familiar with the material of Section 4.

As detailed in Section 4 and Annex B, an implementation of a primitive may make certain assumptions about its inputs, as listed with the specification of each primitive. For example, if EC domain parameters $q$, $a$, $b$, $r$ and $G$ and a public key $W$ are inputs to a primitive, the implementation may generally assume that the domain parameters are valid, i.e., that $G$ has order $r$ on the elliptic curve defined by $a$ and $b$, and $W = s\,G$ for some integer $s$ in the interval $[1, r - 1]$. The behavior of an implementation is unconstrained in the case that $W$ is not a multiple of $G$, and in such a case the implementation may or may not output an error condition. It is up to the properly implemented scheme to ensure that only appropriate inputs are passed to a primitive, or to accept the risks of passing inappropriate inputs. For more on conforming with a primitive, see Annex B.

### 7.2.1 ECSVDP-DH

ECSVDP-DH is Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version. It is based on the work of [DH76], [Mil86], and [Kob87]. This primitive derives a shared secret value from one party's private key and another party's public key, where both have the same set of EC domain parameters. If two parties correctly execute this primitive, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the schemes ECKAS-DH1 and DL/ECKAS-DH2. It assumes that the input keys are valid (see also Section 7.2.2).

**Input:**

— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated with the keys $s$ and $W$ (the domain parameters shall be the same for both $s$ and $W$)
— the party's own private key $s$
— the other party's public key $W$

**Assumptions:** private key $s$, EC domain parameters $q$, $a$, $b$, $r$ and $G$, and public key $W$ are valid; both keys are associated with the domain parameters

**Output:** the derived shared secret value, which is a field element $z \in GF(q)$; or "error"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. Compute an elliptic curve point $P = s\,W$
2. If $P = \mathcal{O}$ output "error" and stop.
3. Let $z = x_P$, the $x$-coordinate of $P$.

4. Output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $G$
— at least one valid private key $s$ for each set of domain parameters
— all valid public keys $W$ associated with the same set of domain parameters as $s$

NOTES

1—This primitive does not address small subgroup attacks (see Annex D.5.1.6), which may occur when the public key $W$ is not valid. To prevent them, a key agreement scheme should validate the public key $W$ before executing this primitive (see also Section 7.2.2).

2—When the public key $W$ and the private key $s$ are valid, $W$ has order $r$ and $s$ is in the interval [1, $r$-1]. Thus the point $P$ in this case is not the element $O$.

### 7.2.2 ECSVDP-DHC

ECSVDP-DHC is Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version with cofactor multiplication. It is based on the work of [DH76], [Mil86], [Kob87], [LMQ98] and [Kal98a]. This primitive derives a shared secret value from one party's private key and another party's public key, where both have the same set of EC domain parameters. If two parties correctly execute this primitive, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the schemes ECKAS-DH1 and DL/ECKAS-DH2. It does not assume the validity of the input public key (see also Section 7.2.1).

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$, $G$ and the cofactor associated with the keys $s$ and $W$ (the domain parameters shall be the same for both $s$ and $W$)
— the party's own private key $s$
— the other party's public key $W$
— an indication as to whether compatibility with ECSVDP-DH is desired

**Assumptions:** private key $s$, EC domain parameters $q$, $a$, $b$, $r$, $G$ and $k$ are valid; the private key is associated with the domain parameters; $W$ is on the elliptic curve defined by $a$ and $b$ over GF ($q$); GCD ($k$, $r$) = 1

**Output:** the derived shared secret value, which is a field element $z \in GF(q)$; or "invalid public key"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. If compatibility with ECSVDP-DH is desired, then compute an integer $t = k^{-1}s \bmod r$; otherwise set $t = s$.
2. Compute an elliptic curve point $P = kt\,W$
3. If $P = O$ output "invalid public key" and stop.
4. Let $z = x_P$, the $x$-coordinate of $P$.
5. Output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$, $G$ and $k$
— at least one valid private key $s$ for each set of domain parameters

— all points $W$ on the curve over GF ($q$) defined by $a$ and $b$, where $q$, $a$ and $b$ are from the domain parameters of $s$
— compatibility with ECSVDP-DH may be preset by the implementation, or given as an input flag

NOTES

1—This primitive addresses small subgroup attacks, which may occur when the public key $W$ is not valid (see Annex D.5.1.6). A key agreement scheme only needs to validate that $W$ is on the elliptic curve defined by $a$ and $b$ over $GF$ ($q$) before executing this primitive (see also Section 7.2.1).

2—The cofactor $k$ depends only on the EC domain parameters. Hence, it can be computed once for a given set of domain parameters and stored as part of the domain parameters. Similarly, in the compatibility case, the value $k^{-1}$ can be computed once and stored with the domain parameters, and the integer $t$ can be computed once for a given private key $s$. Algorithms for computing or verifying the cofactor are included in Annex A.12.3.

3—When the public key $W$ and the private key $s$ are valid, the point $P$ will be an element of a subset of the elliptic curve that consists of all the multiples of $G$ (except for the element $\mathcal{O}$). As a consequence, $z$ will always be defined in this case. When the public key is invalid, the output will be either "invalid public key" or an element of order $r$ on the elliptic curve; in particular, it will not be in a small subgroup.

4—In the compatibility case, ECSVDP-DHC computes the same output for valid keys as ECSVDP-DH, so an implementation that conforms with ECSVDP-DHC in the compatibility case also conforms with ECSVDP-DH.

### 7.2.3 ECSVDP-MQV

ECSVDP-MQV is Elliptic Curve Secret Value Derivation Primitive, Menezes-Qu-Vanstone version. It is based on the work of [LMQ98]. This primitive derives a shared secret value from one party's two key pairs and another party's two public keys, where all the keys and values have the same set of EC domain parameters. If two parties correctly execute this primitive, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the scheme ECKAS-MQV. It assumes that the input keys are valid (see also Section 7.2.4).

In this primitive, let $h = \lceil (\log_2 r) / 2 \rceil$. Note that $h$ depends only on the EC domain parameters, and hence can be computed once for a given set of domain parameters.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated keys $s$, ($u$, $V$), $W$, $V$ (the domain parameters shall be the same for these keys)
— the party's own first private key $s$
— the party's own second key pair ($u$, $V$), where $V = (x, y)$
— the other party's first public key $W$
— the other party's second public key $V = (x, y)$

**Assumptions:** private key $s$, key pair ($u$, $V$), public keys $W$, $V$ and EC domain parameters $q$, $a$, $b$, $r$ and $G$ are valid; all the keys are associated with the domain parameters

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF$ ($q$); or "error"

**Operation.** The shared secret value $z$ shall be computed by the following or an equivalent sequence of steps:

1. Convert $x$ into an integer $i$ using FE2IP.
2. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.

3. Convert $x$ into an integer $i$ using FE2IP.
4. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.
5. Compute an integer $e = ts + u \bmod r$.
6. Compute an elliptic curve point $P = e(V + tW)$.
7. If $P = 0$ output "error" and stop.
8. Let $z = x_P$, the $x$-coordinate of $P$.
9. Output $z$ as the shared secret value.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $g$
— at least one valid private key $s$ for each set of domain parameters
— all valid key pairs $(u, V)$ associated with the same set of domain parameters as $s$
— all valid public keys $w$ and $v$ associated with the same set of domain parameters as $s$

NOTES

1—This primitive does not address small subgroup attacks (see Annex D.5.1.6), which may occur when the public keys $W$ and $V$ are not valid. To prevent them, a key agreement scheme should validate the public keys $W$ and $V$ before executing this primitive (see also Section 7.2.4).

2—When the public keys $W$ and $V$ are valid, the point $P$ will be an element of a subset of the elliptic curve that consists of all the multiples of $G$. It is possible (although very unlikely) that even though all the keys and parameters are valid, $P$ will be the point 0. In this case, the primitive will output "error," and will need to be re-run with a different input. Except for this rare case, $z$ will always be defined as long as the input keys and parameters are valid.

### 7.2.4 ECSVDP-MQVC

ECSVDP-MQVC is Elliptic Curve Secret Value Derivation Primitive, Menezes-Qu-Vanstone version with cofactor multiplication. It is based on the work of [LMQ98] and [Kal98a]. This primitive derives a shared secret value from one party's two key pairs and another party's two public keys, where all the keys and values have the same set of EC domain parameters. If two parties correctly execute this primitive, they will produce the same output. This primitive can be invoked by a scheme to derive a shared secret key; specifically, it may be used with the scheme ECKAS-MQV. It does not assume the validity of the input public keys (see also Section 7.2.3).

In this primitive, let $h = \lceil (\log_2 r) / 2 \rceil$. Note that $h$ depends only on the EC domain parameters, and hence can be computed once for a given set of domain parameters.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$, $G$, and the cofactor $k$ associated with keys $s$, $(u, V)$, $W$, $V$ (the domain parameters shall be the same for these keys)
— the party's own first private key $s$
— the party's own second key pair $(u, V)$, where $V = (x, y)$
— the other party's first public key $W$
— the other party's second public key $V = (x, y)$
— an indication as to whether compatibility with ECSVDP-MQV is desired

**Assumptions:** private key $s$, key pair $(u, V)$, and EC domain parameters $q$, $a$, $b$, $r$, $G$ and $k$ are valid; private key $s$ and key pair $(u, V)$ are associated with the domain parameters; $W$ and $V$ are on the elliptic curve defined by $a$ and $b$ over GF $(q)$; GCD $(k, r) = 1$

**Output:** the derived shared secret value, which is a nonzero field element $z \in GF(q)$; or "invalid public key"

**Operation.** The shared secret value *z* shall be computed by the following or an equivalent sequence of steps:

1. Convert *x* into an integer *i* using FE2IP.
2. Compute an integer $t = i \bmod 2^h$. Let $t = t + 2^h$.
3. Convert $x_U$ into an integer $i_U$ using FE2IP.
4. Compute an integer $t_U = i_U \bmod 2^h$. Let $t_U = t_U + 2^h$.
5. If compatibility with ECSVDP-MQV is desired, the compute an integer $e = k^{-1}(ts + u) \bmod r$; otherwise compute an integer $e = ts + u \bmod r$.
6. Compute an elliptic curve point $P = ke\,(V + t_U W_V)$.
7. If $P = 0$ output "invalid public key" and stop.
8. Let $z = x_P$, the *x*-coordinate of *P*.
9. Output *z* as the shared secret value.

**Conformance region recommendation.** A conformance region should include:

— at least one valid set of EC domain parameters *q*, *a*, *b*, *r*, *g* and *k*
— at least one valid private key *s* for each set of domain parameters
— all valid key pairs (*u*, *V*) associated with the same set of domain parameters as *s*
— all points $W_U$ and $V_U$ on the curve over GF (*q*) defined by *a* and *b*, where *q*, *a* and *b* are from the domain parameters of *s*
— compatibility with ECSVDP-MQV may be preset by the implementation, or given as an input flag

NOTES

1—This primitive addresses small subgroup attacks, which may occur when the public keys $W_U$ and $V_U$ are not valid (see Annex D.5.1.6). A key agreement scheme only needs to validate that $W_U$ and $V_U$ are in GF (*q*) before executing this primitive (see also Section 7.2.3).

2—The cofactor *k* depends only on the EC domain parameters. Hence, it can be computed once for a given set of domain parameters and stored as part of the domain parameters. Similarly, in the compatibility case, the value $k^{-1}$ can be computed once and stored with the domain parameters. An equivalent way to compute the integer *e* in this case is as $(tk^{-1}s + k^{-1}u) \bmod r$, where $k^{-1}s \bmod r$ and $k^{-1}u \bmod r$ can be computed once for each given private key *s* and *u*. Algorithms for computing or verifying the cofactor are included in Annex A.12.3.

3—When the public key $W_U$ is valid, the point *P* will be an element of a subset of the elliptic curve that consists of all the multiples of *G*. It is possible (although very unlikely) that even though all the keys and parameters are valid, *P* will be the point 0, and the primitive will output "invalid public key." In this case, the primitive will need to be re-run with a different input. Except for this rare case, *z* will always be defined as long as the input keys and parameters are valid. When one of the public keys is invalid, the output will be either "invalid public key" or the *x* coordinate of a point of order *r* on the elliptic curve; in particular, it will not be the *x* coordinate of a point in a small subgroup.

4—In the compatibility case, ECSVDP-MQVC computes the same output for valid keys as ECSVDP-MQV, so an implementation that conforms with ECSVDP-MQVC in the compatibility case also conforms with ECSVDP-MQV.

### 7.2.5 ECSP-NR

ECSP-NR is Elliptic Curve Signature Primitive, Nyberg-Rueppel version. It is based on the work of [Mil86], [Kob87] and [NR93]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the ECVP-NR primitive. Note, however, that no EC signature schemes with message recovery are defined in this version of the standard (see Annex C.3.4). ECSP-NR may also be used in a signature scheme with appendix and can be invoked in the scheme DLSSA as part of signature generation.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated with the key $s$
— the signer's private key $s$
— the message representative, which is an integer $f$ such that $0 \le f < r$

**Assumptions:** private key $s$ and EC domain parameters $q$, $a$, $b$, $r$ and $G$ are valid and associated with each other; $0 \le f < r$

**Output:** the signature, which is a pair of integers $(c, d)$, where $1 \le c < r$ and $0 \le d < r$

**Operation.** The signature $(c, d)$ shall be computed by the following or an equivalent sequence of steps:

1. Generate a key pair $(u, V)$ with the same set of domain parameters as the private key $s$. (See the note below.) Let $V = (x_V, y_V)$ ($V \neq \mathbf{0}$ because $V$ is a public key).
2. Convert $x_V$ into an integer $i$ with primitive FE2IP (recall that $x_V$ is an element of $GF(q)$).
3. Compute an integer $c = i + f \bmod r$. If $c = 0$, then go to Step 1.
4. Compute an integer $d = u - sc \bmod r$.
5. Output the pair $(c, d)$ as the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $G$
— at least one valid private key $s$ for each set of domain parameters
— all message representatives $f$ in the range $[0, r - 1]$, where $r$ is from the domain parameters of $s$

NOTE—The key pair in Step 1 should be a one-time key pair which is generated and stored by the signer following the security recommendations of D.3.2, D.4.2.2, D.6 and D.7. A new key pair should be generated for every signature. The one-time private key $u$ should be discarded after step 4, as its recovery by an opponent can lead to the recovery of the private key $s$.

### 7.2.6 ECVP-NR

ECVP-NR is Elliptic Curve Verification Primitive, Nyberg-Rueppel version. It is based on the work of [Mil86], [Kob87] and [NR93]. This primitive recovers the message representative that was signed with ECSP-NR, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that no EC signature schemes with message recovery are defined in this version of the standard (see Annex C.3.4). ECVP-NR may also be used in a signature scheme with appendix and can be invoked in the scheme DLSSA as part of signature verification.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated with the key $W$
— the signer's public key $W$
— the signature to be verified, which is a pair of integers $(c, d)$

**Assumptions:** public key $W$ and EC domain parameters $q$, $a$, $b$, $r$ and $G$ are valid and associated with each other

**Output:** the message representative, which is an integer $f$ such that $0 \le f < r$; or "invalid"

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1. If $c$ is not in $[1, r - 1]$ or $d$ is not in $[0, r - 1]$, output "invalid" and stop.

2. Compute an elliptic curve point $P = d\,G + c\,W$. If $P = \mathcal{O}$, output "invalid" and stop. Otherwise, $P = (x_P, y_P)$.
3. Convert the field element $x_P$ to an integer $i$ with primitive FE2IP.
4. Compute an integer $f = c - i \bmod r$.
5. Output $f$ as the message representative.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $G$
— at least one valid public key $W$ for each set of domain parameters
— all purported signatures $(c, d)$ that can be input to the implementation; this should at least include all $(c, d)$ such that $c$ and $d$ are in the range $[0, r - 1]$ , where $r$ is from the domain parameters of $W$

### 7.2.7 ECSP-DSA

ECSP-DSA is Elliptic Curve Signature Primitive, DSA version. It is based on the work of [Mil86], [Kob87] and [Kra93]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer. The message representative cannot be recovered from the signature, but ECVP-DSA can be used in the scheme ECSSA to verify the signature.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated with the key $s$
— the signer's private key $s$
— the message representative, which is an integer $f \geq 0$

**Assumptions:** private key $s$ and EC domain parameters $q$, $a$, $b$, $r$ and $G$ are valid and associated with each other; $f \geq 0$

**Output:** the signature, which is a pair of integers $(c, d)$, where $1 \leq c < r$ and $1 \leq d < r$

**Operation.** The signature $(c, d)$ shall be computed by the following or an equivalent sequence of steps:

1. Generate a key pair $(u, V)$ with the same set of domain parameters as the private key $s$. (See the note below.) Let $V = (x_V, y_V)$ ($V \neq \mathcal{O}$ because $V$ is a public key).
2. Convert $x_V$ into an integer $i$ with primitive FE2IP (recall that $x_V$ is an element of $GF(q)$).
3. Compute an integer $c = i \bmod r$. If $c = 0$, then go to Step 1.
4. Compute an integer $d = u^{-1}(f + sc) \bmod r$. If $d = 0$, then go to Step 1.
5. Output the pair $(c, d)$ as the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $G$
— at least one valid private key $s$ for each set of domain parameters
— a range of message representatives $f$; this should at least include all $f \geq 0$ with bit length no greater than that of $r$, where $r$ is from the domain parameters of $s$

NOTE—The key pair in Step 1 should be a one-time key pair which is generated and stored by the signer following the security recommendations of D.3.2, D.4.2.2, D.6 and D.7. A new key pair should be generated for every signature. The one-time private key $u$ should be discarded after step 4, as its recovery by an opponent can lead to the recovery of the private key $s$.

### 7.2.8 ECVP-DSA

ECVP-DSA is Elliptic Curve Verification Primitive, DSA version. It is based on the work of [Mil86], [Kob87] and [Kra93]. This primitive verifies whether the message representative and the signature are

consistent given the key and the domain parameters. It can be invoked in the scheme ECSSA as part of signature verification.

**Input:**
— the EC domain parameters $q$, $a$, $b$, $r$ and $G$ associated with the key $W$
— the signer's public key $W$
— the message representative, which is an integer $f \geq 0$
— the signature to be verified, which is a pair of integers $(c, d)$

**Assumptions:** public key $W$ and EC domain parameters $q$, $a$, $b$, $r$ and $G$ are valid and associated with each other; $f \geq 0$

**Output:** "valid" if $f$ and $(c, d)$ are consistent given the key and the domain parameters; "invalid" otherwise

**Operation.** The output shall be computed by the following or an equivalent sequence of steps:

1. If $c$ is not in $[1, r-1]$ or $d$ is not in $[1, r-1]$, output "invalid" and stop.
2. Compute integers $h = d^{-1} \bmod r$; $h_1 = f\,h \bmod r$; $h_2 = c\,h \bmod r$.
3. Compute an elliptic curve point $P = h_1G + h_2W$. If $P = \mathcal{O}$, output "invalid" and stop. Otherwise, $P = (x_P, y_P)$.
4. Convert the field element $x_P$ to an integer $i$ with primitive FE2IP.
5. Compute an integer $c' = i \bmod r$.
6. If $c' = c$, then output "valid"; else, output "invalid."

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of EC domain parameters $q$, $a$, $b$, $r$ and $G$
— at least one valid public key $W$ for each set of domain parameters
— all message representatives $f \geq 0$ that can be input to the implementation; this should at least include all $f$ with bit length no greater than that of $r$, where $r$ is from the domain parameters of $W$
— all purported signatures $(c, d)$ that can be input to the implementation; this should at least include all $(c, d)$ such that $c$ and $d$ are in the range $[1, r-1]$ , where $r$ is from the domain parameters of $W$

# 8. Primitives Based on the Integer Factorization Problem

This section specifies the family of cryptographic primitives based on the integer factorization problem, also known as the IF family. For background information on this family, see [RSA78], Annex A.1.3 and A.1.4.

There are two types of primitives in this family. While they are largely similar, they are based on slightly different keys and operations. One type is known as RSA (for "Rivest-Shamir-Adleman"—see [RSA78]); the other is known as RW (for "Rabin-Williams"—see [Rab79], [Wil80]).

## 8.1 The IF Setting

### 8.1.1 Notation

The list below introduces the notation used in Sections 8, 10 and 11. It is meant as a reference guide only; for complete definitions of the terms listed, refer to the appropriate text. Some other symbols are also used occasionally; they are introduced in the text where appropriate.

| | |
|---|---|
| $n$ | The modulus, part of the keys |
| $p$, $q$ | The two primes whose product is the modulus $n$ |
| $e$ | The public exponent, part of a public key ($e$ is odd for RSA; $e$ is even for RW) |
| $d$ | The private exponent, part of some of the representations of a private key |
| $d_1$ | $d \bmod (p-1)$, part of one of the representations of a private key |
| $d_2$ | $d \bmod (q-1)$, part of one of the representations of a private key |
| $c$ | $q^{-1} \bmod p$, part of one of the representations of a private key |
| $(n, e)$ | An RSA or RW public key |
| $K$ | An RSA or RW private key |
| $s$ | Signature, an integer, computed by a signature primitive |
| $g$ | Encrypted message, an integer, computed by an encryption primitive |
| $f$ | Message representative, an integer, computed by a message encoding operation |
| $M$ | The message, an octet string, which is encrypted in an encryption scheme or whose signature is computed by a signature scheme |

NOTE—Multiplication of integers is denoted by $\times$, although this symbol may be omitted when such omission does not cause ambiguity. When typographically convenient, notation such as $\exp(f, e)$, where $f$ and $e$ are integers, will be used to denote raising $f$ to the power $e$. The more traditional notation $f^e$ may also be used

### 8.1.2 Domain Parameters in the IF Family

Unlike the DL or EC families, the IF family has no domain parameters. The keys contain all the necessary information within themselves. While DL or EC primitives and schemes sometimes require domain parameters to be common to a group of keys, no similar requirement exists for any IF primitive or scheme. Of course, as with other families, parties need to be aware of each other's capabilities in order to work together—for example, some parties may have restrictions on the size of the modulus they can use or generate, while others may only work with a particular fixed public exponent (see below).

### 8.1.3 Keys in the IF Family

Since there are two types of IF primitives, there are also two types of IF keys. They are similar, but the distinctions between them are significant.

#### 8.1.3.1 RSA Key Pairs

An RSA *public key* consists of a *modulus n*, which is the product of two odd positive prime integers $p$ and $q$, and an odd *public exponent* $e$ ($3 \le e < n$) which is an integer relatively prime to ($p - 1$) and ($q - 1$). The corresponding RSA *private exponent* is an integer $d$ ($1 \le d < n$) such that

$$d\,e \equiv 1 \pmod{\text{LCM}(p - 1, q - 1)}.$$

Note that there may be more than one private exponent $d$ corresponding to a public key ($n$, $e$).

An RSA *private key K* may have multiple representations. The use of one of the following three representations is recommended in this standard:

1. a pair ($n$, $d$)
2. a quintuple ($p$, $q$, $d_1$, $d_2$, $c$), where $d_1 = d \bmod (p - 1)$, $d_2 = d \bmod (q - 1)$, and $c = q^{-1} \bmod p$
3. a triple ($p$, $q$, $d$)

#### 8.1.3.2 RW Key Pairs

An RW *public key* consists of a *modulus n*, which is the product of two odd positive prime integers $p$ and $q$ such that $p \not\equiv q \pmod 8$, and an even *public exponent* $e$ ($2 \le e < n$) which is an integer relatively prime to $(p - 1)(q - 1)/4$. (Note that these conditions imply that $p \equiv q \equiv 3 \pmod 4$; moreover, one of the primes is congruent to 3 (mod 8) and the other is congruent to 7 (mod 8).) The corresponding RW *private exponent* is an integer $d$ ($1 \le d < n$) such that

$$d\,e \equiv 1 \pmod{\text{LCM}(p - 1, q - 1) / 2}.$$

Note that there may be more than one private exponent $d$ corresponding to a public key ($n$, $e$).

An RW *private key K* may have multiple representations. The use of one of the following three representations is recommended in this standard:

1. a pair ($n$, $d$)
2. a quintuple ($p$, $q$, $d_1$, $d_2$, $c$), where $d_1 = d \bmod (p - 1)$, $d_2 = d \bmod (q - 1)$, and $c = q^{-1} \bmod p$
3. a triple ($p$, $q$, $d$)

#### 8.1.3.3 Considerations Common to RSA and RW Key Pairs

An RSA or RW key pair may or may not be generated by the party using it, depending on the trust model. This and other security issues related to RSA and RW key pair generation are discussed in D.3.2 and D.4.3. A suggested method for generating RSA key pairs is contained in Annex A.16.11. A suggested method for generating RW key pairs is contained in Annex A.16.12.

Parties establish RSA and RW keys as part of key management for a scheme. Depending on the key management technique, it is possible that an established key does not satisfy the intended definition for the key, even though it has the same general form (e.g., components $n$ and $e$). To accommodate this possibility, the terms "RSA public key," "RW public key," "RSA private key," and "RW private key" shall be understood in this standard as referring to instances of the general form for the key. The terms *valid RSA public key*, *valid RW public key*, *valid RSA private key*, *valid RW private key*, *valid RSA key pair* and *valid RW key pair* shall be reserved for keys satisfying the definitions.

*Key validation* is the process of determining whether a key is valid. Further discussion of key validation is contained in D.3.3, although no algorithm for IF public key validation is suggested in Annex A. No

algorithm is given for IF private key validation, because, generally, a party controls its own private key and need not validate it. However, private key validation may be performed if desired.

For security, the primes $p$ and $q$ need to be generated so that they are unpredictable and inaccessible to an adversary. Whatever form a private key is represented in, it needs to be stored so that it is inaccessible to an adversary. The compromise of $p$, $q$, $d$, $d_1$, $d_2$, or $c$ will aid the adversary in recovering the private key. These values should be discarded if not used.

Three representations are provided for IF private keys because of performance tradeoffs between the representations. Use of the quintuple representation tends to result in faster performance for larger sizes of $n$.

## 8.2 Primitives

This section describes primitives used in the IF family. Before proceeding with this section, the reader should be familiar with the material of Section 4.

As detailed in Section 4 and Annex B, an implementation of a primitive may make certain assumptions about its inputs, as listed with the specification of the primitive. For example, if an RSA public key $(n, e)$ is an input to a primitive, an implementation may assume that the public key is valid, i.e., that $n$ is a product of two odd primes, $3 \le e < n$, and $e$ is relatively prime to $(p - 1)(q - 1)$. The behavior of an implementation is not specified in the case that the input does not satisfy these conditions, and in such a case the implementation may or may not output an error condition. It is up to the properly implemented scheme to ensure that only appropriate inputs are passed to a primitive, or to accept the risks of passing inappropriate inputs. For more on conforming with a primitive, see Annex B.

The primitives described in this section can be combined into four pairs: message representatives encrypted with IFEP-RSA can be decrypted with IFDP-RSA; message representatives signed with IFSP-RSA1, IFSP-RSA2 or IFSP-RW can be recovered with IFVP-RSA1, IFVP-RSA2 or IFVP-RW, respectively. While IFEP-RSA is mathematically identical to IFVP-RSA1, and IFDP-RSA is mathematically identical to IFSP-RSA1, these primitives are used for entirely different purposes and should not be confused. The three signature primitives are similar, but have some important differences (see Annex C.3.6.) To aid in defining these primitives, the operation "IF Private Key Operation" is defined first.

### 8.2.1 IF Private Key Operation

IF Private Key Operation is not a primitive in itself, but rather is used in some of the primitives below. The operation produces the same result independent of the representation of the private key.

**Input:**

- an RSA or RW private key $K$ represented as one of the following
  - a pair $(n, d)$
  - a quintuple $(p, q, d_1, d_2, c)$
  - a triple $(p, q, d)$
- an integer $i$ such that $0 \le i < n$ (where $n = pq$ if the second or the third representation of $K$ is used), to which the private key operation is to be applied

**Assumptions:** private key $K$ is valid; $0 \le i < n$

**Output:** an integer $j$ such that $0 \le j < n$, the result of the private key operation on $i$

**Operation.** The integer $j$ shall be computed by the following or an equivalent sequence of steps:

I. If the first representation $(n, d)$ of $K$ is used:
   1. Let $j = \exp(i, d) \bmod n$.
   2. Output $j$.

II. If the second representation $(p, q, d_1, d_2, c)$ of $K$ is used:
   1. Let $j_1 = \exp(i, d_1) \bmod p$.
   2. Let $j_2 = \exp(i, d_2) \bmod q$.
   3. Let $h = c\,(j_1 - j_2) \bmod p$.
   4. Let $j = j_2 + h\,q$.
   5. Output $j$.

III. If the third representation $(p, q, d)$ of $K$ is used:
   1. Let $j_1 = \exp(i, d \bmod (p - 1)) \bmod p$.
   2. Let $j_2 = \exp(i, d \bmod (q - 1)) \bmod q$.
   3. Let $h = q^{-1} \times (j_1 - j_2) \bmod p$.
   4. Let $j = j_2 + h\,q$.
   5. Output $j$.

### 8.2.2 IFEP-RSA

IFEP-RSA is RSA Encryption Primitive. It is based on the work of [RSA78]. It is invoked in the scheme IFES as part of encrypting a message, given the message and the public key of the intended recipient. The message can be decrypted within a scheme by invoking IFDP-RSA.

**Input:**
— the recipient's RSA public key $(n, e)$
— the message representative, which is an integer $f$ such that $0 \le f < n$

**Assumptions:** public key $(n, e)$ is valid; $0 \le f < n$

**Output:** the encrypted message representative, which is an integer $g$ such that $0 \le g < n$

**Operation.** The encrypted message representative $g$ shall be computed by the following or an equivalent sequence of steps:

1. Let $g = \exp(f, e) \bmod n$.
2. Output $g$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA public key $(n, e)$
— all message representatives $f$ in the range $[0, n - 1]$

### 8.2.3 IFDP-RSA

IFDP-RSA is RSA Decryption Primitive. It is based on the work of [RSA78]. It is used in the scheme IFES as part of decrypting a message encrypted with the use of IFEP-RSA, given the encrypted message representative and the private key of the recipient.

**Input:**
— the recipient's RSA private key $K$
— the encrypted message representative, which is an integer $g$ such that $0 \le g < n$

**Assumptions:** private key $K$ is valid; $0 \le g < n$

**Output:** the message representative, which is an integer $f$ such that $0 \le f < n$

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1. Perform the IF Private Key Operation described in 8.2.1 on $K$ and $g$ to produce an integer $f$.
2. Output $f$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA private key $K$
— all encrypted message representatives $g$ in the range $[0, n - 1]$, where $n$ is from the private key $K$

### 8.2.4 IFSP-RSA1

IFSP-RSA1 is RSA Signature Primitive, version 1. It is based on the work of [RSA78]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RSA1 primitive. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex C.3.7). IFSP-RSA1 may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature generation.

**Input:**
— the signer's RSA private key $K$
— the message representative, which is an integer $f$ such that $0 \le f < n$

**Assumptions:** private key $K$ is valid; $0 \le f < n$

**Output:** the signature, which is an integer $s$ such that $0 \le s < n$

**Operation.** The signature $s$ shall be computed by the following or an equivalent sequence of steps:

1. Perform the IF Private Key Operation described in 8.2.1 on $K$ and $f$ to produce an integer $s$.
2. Output $s$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA private key $K$
— all message representatives $f$ in the range $[0, n - 1]$, where $n$ is from the private key $K$

### 8.2.5 IFVP-RSA1

IFVP-RSA1 is RSA Verification Primitive, version 1. It is based on the work of [RSA78]. This primitive recovers the message representative that was signed with IFSP-RSA1, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex C.3.7). IFVP-RSA1 may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature verification.

**Input:**
— the signer's RSA public key $(n, e)$
— the signature to be verified, which is an integer $s$

**Assumptions:** public key ($n$, $e$) is valid

**Output:** the message representative, which is an integer $f$ such that $0 \le f < n$; or "invalid"

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1 If $s$ is not in [0, $n - 1$], output "invalid" and stop.
2. Let $f = \exp(s, e) \bmod n$.
3. Output $f$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA public key ($n$, $e$)
— all purported signatures $s$ that can be input to the implementation; this should at least include all $s$ in the range [0, $n - 1$]

### 8.2.6 IFSP-RSA2

IFSP-RSA2 is RSA Signature Primitive, version 2. It is based on the work of [RSA78] and [ISO91]. Its output is at least one bit shorter than the RSA modulus $n$. It can be invoked in a scheme to compute a signature on a message representative of a certain form with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RSA2 primitive. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex C.3.7). IFSP-RSA2 may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature generation.

**Input:**
— the signer's RSA private key $K$
— the message representative, which is an integer $f$ such that $0 \le f < n$, and $f \equiv 12 \pmod{16}$

**Assumptions:** private key $K$ is valid, $0 \le f < n$, and $f$ is congruent to 12 modulo 16

**Output:** the signature, which is an integer $s$ such that $0 \le s < n/2$

**Operation.** The signature $s$ shall be computed by the following or an equivalent sequence of steps:

1. Perform the IF Private Key Operation described in 8.2.1 on $K$ and $f$ to produce an integer $t$.
2. Let $s = \min(t, n - t)$.
3. Output $s$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA private key $K$
— all message representatives $f$ in the range [0, $n - 1$] such that $f \equiv 12 \pmod{16}$, where $n$ is from the private key $K$

### 8.2.7 IFVP-RSA2

IFVP-RSA2 is RSA Verification Primitive, version 2. It is based on the work of [RSA78] and [ISO91]. This primitive recovers the message representative that was signed with IFSP-RSA2, given only the signature, the public key of the signer, and the last four bits of the message representative. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex

C.3.7). IFVP-RSA2 may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature verification.

**Input:**
— the signer's RSA public key $(n, e)$
— the signature to be verified, which is an integer $s$

**Assumption:** public key $(n, e)$ is valid

**Output:** the message representative, which is an integer $f$ such that $0 \le f < n$ and $f \equiv 12 \pmod{16}$; or "invalid"

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1 If $s$ is not in $[0, (n-1)/2]$, output "invalid" and stop.
2. Compute $t = \exp(s, e) \bmod n$.
3. If $t \equiv 12 \pmod{16}$, then let $f = t$.
4. Else let $f = n - t$. If $f \not\equiv 12 \pmod{16}$, output "invalid" and stop.
5. Output $f$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA public key $(n, e)$
— all purported signatures $s$ that can be input to the implementation; this should at least include all $s$ in the range $[0, (n-1)/2]$

### 8.2.8 IFSP-RW

IFSP-RW is RW Signature Primitive. It is based on the work of [Rab79], [Wil80] and [ISO91]. Its output is at least one bit shorter than the RW modulus $n$. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RW primitive. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex C.3.7). IFSP-RW may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature generation.

**Input:**
— the recipient's RW private key $K$
— the message representative, which is an integer $f$ such that $0 \le f < n$, and $f \equiv 12 \pmod{16}$

**Assumptions:** private key $K$ is valid, $0 \le f < n$, and $f$ is congruent to 12 modulo 16

**Output:** the signature, which is an integer $s$ such that $0 \le s < n/2$

**Operation.** The signature $s$ shall be computed by the following or an equivalent sequence of steps:

1. If the Jacobi symbol $\left(\frac{f}{n}\right) = +1$, let $u = f$.
2. Else let $u = f / 2$ (recall that $f$ is even).
3. Perform the IF Private Key Operation described in 8.2.1 on $K$ and $u$ to produce an integer $t$.
4. Let $s = \min(t, n - t)$.
5. Output $s$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RW private key $K$
— all message representatives $f$ in the range $[0, n - 1]$ such that $f \equiv 12 \pmod{16}$, where $n$ is from the private key $K$

NOTE—See Annex A.2.9 for a potentially more efficient implementation of the primitive and A.1.4 and A.2.3 for evaluating Jacobi symbols.

### 8.2.9 IFVP-RW

IFVP-RW is RW Verification Primitive. It is based on the work of [Rab79], [Wil80] and [ISO91]. This primitive recovers the message representative that was signed with IFSP-RW, given only the signature, public key of the signer, and the last four bits of the message representative. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that no IF signature schemes with message recovery are defined in this version of the standard (see Annex C.3.7). IFVP-RW may also be used in a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature verification.

**Input:**
— the signer's RW public key $(n, e)$
— the signature to be verified, which is an integer $s$

**Assumption:** public key $(n, e)$ is valid

**Output:** the message representative, which is an integer $f$ such that $0 \le f < n$ and $f \equiv 12 \pmod{16}$; or "invalid"

**Operation.** The message representative $f$ shall be computed by the following or an equivalent sequence of steps.

1 If $s$ is not in $[0, (n - 1) / 2]$, output "invalid" and stop.
2. Compute $t_1 = \exp(s, e) \bmod n$.
3. Compute $t_2 = n - t_1$.
4. If $t_1 \equiv 12 \pmod{16}$, then let $f = t_1$.
5. Else if $t_1 \equiv 6 \pmod{8}$, then let $f = 2t_1$.
6. Else if $t_2 \equiv 12 \pmod{16}$, then let $f = t_2$.
7. Else if $t_2 \equiv 6 \pmod{8}$, then let $f = 2t_2$.
8. Else output "invalid" and stop.
9. Output $f$.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RW public key $(n, e)$
— all purported signatures $s$ that can be input to the implementation; this should at least include all $s$ in the range $[0, (n - 1)/2]$

# 9. Key Agreement Schemes

The general model for key agreement schemes is given in Section 9.1. Three specific schemes and their allowable options are given in Sections 9.2, 9.3, and 9.4.

## 9.1 General Model

In a key agreement scheme, each party combines its own private key(s) with the other party's public key(s) to come up with a secret key. Other (public or private) information known to both parties may also enter the scheme as *key derivation parameters*. If the parties use the corresponding keys and identical key derivation parameters, and the scheme is executed correctly, the parties will arrive at the same secret key (see note 1, below). A key agreement scheme can allow two parties to derive shared secret keys without any prior shared secret.

A key agreement scheme consists of a key agreement operation, along with supporting key management. Domain parameter and key pair generation for the key agreement schemes are specified further in connection with the DL and EC families (Sections 6 and 7, respectively). Security considerations for key agreement schemes are given in Annex D.5.1.

A key agreement operation has the following form for all the schemes:

1. Establish one or more sets of valid domain parameters with which the parties' key pairs shall be associated.
2. Select one or more valid private keys (and, in some cases, their corresponding public keys) for the operation, associated with the domain parameters established in Step 1. (Note 2.)
3. Obtain one or more other party's purported public keys for the operation. (Note 3.)
4. *(Optional.)* Depending on the cryptographic operations in Step 5, choose an appropriate method to validate the public keys and the domain parameters. If any validation fails, output "invalid" and stop. (Note 4.)
5. Apply certain cryptographic operations to the private and public keys to produce a shared secret value.
6. For each shared secret key to be agreed on, establish or agree on key derivation parameters and derive a shared secret key from the shared secret value and the key derivation parameters using a key derivation function. (Note 5.)

NOTES

1—*(Key confirmation.)* By the definition of a key agreement scheme, if the correct keys are used and computation is performed properly, the shared secret keys computed by the two parties will also be the same. However, to verify the identities of the parties and to ensure that they indeed possess the same key, the parties may need to perform a key confirmation protocol. See Annex D.5.1.3 for more information.

2—*(Repeated use of a key pair.)* A given public/private key pair may be used by either party for any number of key agreement operations, depending on the implementation.

3—*(Authentication of ownership.)* The process of obtaining the other party's public key (Step 3) may involve authentication of ownership of the public key, as further described in D.3.1 and D.5.1.5. This may be achieved by verifying a certificate or by other means. The means by which the key (or the certificate containing it) is obtained may vary, and may include one party sending the key to the other, or a party obtaining the other party's key from a third party or from a local database.

4—*(Domain parameter and key validation.)* Since a key agreement primitive assumes (with some exceptions, depending on the primitive chosen) that the domain parameters and the public key are valid, the result of the primitive is undefined otherwise. Consequently, it is recommended that parties validate the set(s) of domain parameters in step 1 and the public key(s) in step 4, unless the risk of operating on invalid domain parameters or keys is mitigated by other means, as discussed further in D.3.2 and D.5.1.6. Examples of "other means" include validation within the supporting key management, or use of -DHC and -MQVC primitives together with simpler validation. In the case of key agreement it is only necessary that each party is assured of domain parameter and public key validity prior to use of the shared secret key. Therefore, while it may be more efficient to validate domain parameters once and then validate each key as it is used, there may be instances where domain parameter validation follows public key validation. In other words, an implementation may perform domain parameter validation and public key validation in a different order than assumed in the specification (step 1 followed by step 4), with equivalent effect, provided that both occur prior to use of the shared secret key.

5—*(Key derivation parameters.)* Depending on the key derivation function, there may be security-related constraints on the set of allowed key derivation parameters. The interpretation of these parameters is left to the implementation. For instance, it may contain key-specific information, protocol-related public information, and supplementary, private information. For security, the interpretation should be unambiguous. See Annex D.5.1.4 for further discussion.

6—*(Attributes of the shared secret key.)* The attributes of the shared secret key depend on the particular key agreement scheme used, the attributes of the public/private key pairs, the nature of the parameters to the key derivation function, and whether or not key confirmation is performed. See Annex D.5.1 for further discussion of the attributes of the shared secret key.

7—*(Error conditions.)* The two parties may produce errors under certain conditions, such as the following:

— private key not found in step 2
— public key not found in step 3
— public key not valid in step 4
— private key or public key not supported in step 5
— key derivation parameter not supported in step 6

Such error conditions should be detected and handled appropriately by an implementation, but the specific methods for detecting and handling them are outside of the scope of this standard.

## 9.2 DL/ECKAS-DH1

DL/ECKAS-DH1 is Discrete Logarithm and Elliptic Curve Key Agreement Scheme, Diffie-Hellman version, where each party contributes one key pair.

### 9.2.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme:

— a secret value derivation primitive, which shall be:
  — DLSVDP-DH, DLSVDP-DHC, ECSVDP-DH or ECSVDP-DHC
— for a -DHC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -DH primitive is desired
— a key derivation function, which should be KDF1 (Section 13.1), or a function designated for use with DL/ECKAS-DH1 in an addendum to this standard

The above information may remain the same for any number of executions of the key agreement scheme, or it may be changed at some frequency. The information need not be kept secret.

### 9.2.2 Key Agreement Operation

A sequence of shared secret keys $K_1$, $K_2$, …, $K_t$ shall be generated by each party by performing the following or an equivalent sequence of steps:

1. Establish the valid set of DL or EC domain parameters with which the parties' key pairs shall be associated.
2. Select a valid private key *s* for the operation, associated with the parameters established in Step 1.
3. Obtain the other party's purported public key *w* ¢for the operation, associated with the parameters established in Step 1.
4. *(Optional.)* If the selected secret value derivation primitive is DLSVDP-DHC or ECSVDP-DHC, then validate that *w* ¢is an element in the appropriate group (i.e., in GF (*q*) for DL or on the elliptic curve for EC—see Sections 6.2.2 and 7.2.2); otherwise validate that *w* ¢is a valid public key. If any validation fails, output "invalid public key" and stop.
5. Compute a shared secret value *z* from the private key *s* and the other party's public key *w* ¢with the selected secret value derivation primitive (see Section 9.2.1).
6. Convert the shared secret value *z* to an octet string *Z* using FE2OSP.
7. For each shared secret key to be agreed on:
   a. Establish or otherwise agree on key derivation parameters $P_i$ for the key.
   b. Derive a shared secret key $K_i$ from the octet string *Z* and the key derivation parameters $P_i$ with the selected key derivation function (see Section 9.2.1).

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of domain parameters
— at least one valid private key *s* for each set of domain parameters
— all valid public keys *w* ¢associated with the same set of domain parameters as *s*; if key validation is performed or a -DHC primitive is used, invalid public keys that are appropriately handled by the implementation may also be included in the conformance region
— a range of key derivation parameters *P*

NOTE—These schemes, with appropriate restrictions on the scheme options and inputs, may be compatible with a techniques in ANSI X9.42 [ANS98b] (in the DL case) and ANSI X9.63 [ANS98f] (in the EC case).

## 9.3 DL/ECKAS-DH2

DL/ECKAS-DH2 is the Discrete Logarithm and Elliptic Curve Key Agreement Scheme, Diffie-Hellman version, where each party contributes two key pairs. Particular variants may be derived from the work of [MTI86], [Gos90], [Gun90], [ANS98b], [Joh] and [BJM98].

### 9.3.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme:

— two secret value derivation primitives, each of which (independently) shall be:
  — DLSVDP-DH, DLSVDP-DHC, ECSVDP-DH or ECSVDP-DHC
— for each -DHC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -DH primitive is desired
— if the two secret value derivation primitives are the same (and have the same compatibility option in the -DHC case), an indication whether compatibility with DL/ECKAS-DH1 is desired
— a key derivation function, which should be KDF1 (Section 13.1), or a function designated for use with DL/ECKAS-DH2 in an addendum to this standard

The above information may remain the same for any number of executions of the key agreement function, or it may be changed at some frequency. The information need not be kept secret.

### 9.3.2 Key Agreement Operation

A sequence of shared secret keys $K_1$, $K_2$, …, $K_t$ shall be generated by each party by performing the following or an equivalent sequence of steps:

1. Establish the valid set of DL or EC domain parameters with which the parties' first key pairs shall be associated.
2. Establish the valid set of DL or EC domain parameters with which the parties' second key pairs shall be associated.
3. Select valid first and second private keys *s* and *u* for the operation, associated with the domain parameters established in steps 1 and 2, respectively.
4. Obtain the other party's first and second purported public keys *w*¢and *v*¢for the operation, associated with the domain parameters established in Steps 1 and 2, respectively.
5. *(Optional.)* If the first selected secret value derivation primitive is DLSVDP-DHC or ECSVDP-DHC, then validate that *w*¢is an element in the appropriate group (i.e., in GF(*q*) for DL or on the elliptic curve for EC—see Sections 6.2.2 and 7.2.2); otherwise validate that *w*¢is a valid public key. If the second selected secret value derivation primitive is DLSVDP-DHC or ECSDVP-DHC, then validate that *v*¢is an element in the appropriate group; otherwise validate that *v*¢is a valid public key. If any validation fails, output "invalid public key" and stop.
6. Compute a first shared secret value $z_1$ from *s* and *w*¢with the first selected secret value derivation primitive (see Section 9.3.1). Convert $z_1$ to an octet string $Z_1$ using FE2OSP.
7. Compute a second shared secret value $z_2$ from *u* and *v*¢with the second selected secret value derivation primitive (see Section 9.3.1). Convert $z_2$ to an octet string $Z_2$ using FE2OSP.
8. If compatibility with DL/ECKAS-DH1 is desired, the selected private keys *s* and *u* (and associated domain parameters) are the same, and the other party's public keys *w*¢and *v*¢(and associated domain parameters) are the same, then let $Z = Z_1$. Otherwise, let $Z = Z_1 \| Z_2$.
9. For each shared secret key to be agreed on:
   a. Establish or otherwise agree on key derivation parameters $P_i$ for the key.
   b. Derive a shared secret key $K_i$ from the octet string *Z* and the key derivation parameters $P_i$ with the selected key derivation function (see Section 9.3.1).

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of domain parameters for the first pair of keys
— at least one valid private key *s* for each set of domain parameters for the first pair of keys
— all valid public keys *w*¢associated with the same set of domain parameters as *s*
— at least one valid set of domain parameters for the second pair of keys
— at least one valid private key *u* for each set of domain parameters for the second pair of keys
— all valid public keys *v*¢associated with the same set of domain parameters as *u*
— a range of key derivation parameters *P*
— if key validation is performed or a -DHC primitive is used, invalid public keys that are appropriately handled by the implementation may also be included in the conformance region

NOTES

1—If the two secret value derivation primitives are the same, compatibility with DL/ECKAS-DH1 is selected, and each party contributes two identical key pairs, then DL/ECKAS-DH2 will compute the same output as DL/ECKAS-DH1 with the same primitive and key pairs.

2—This scheme, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.42 [ANS98b] (in the DL case) and ANSI X9.63 [ANS98f] (in the EC case).

## 9.4 DL/ECKAS-MQV

DL/ECKAS-MQV is Discrete Logarithm and Elliptic Curve Key Agreement Scheme, Menezes-Qu-Vanstone version. Each party contributes two key pairs.

### 9.4.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme:

— a secret value derivation primitive, which shall be
  — DLSVDP-MQV, DLSVDP-MQVC, ECSVDP-MQV, or ECSVDP-MQVC
— for an -MQVC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -MQV primitive is desired
— a key derivation function, which should be KDF1 (Section 13.1), or a function designated for use with DL/ECKAS-MQV in an addendum to this standard

The above information may remain the same for any number of executions of the key agreement scheme, or it may be changed at some frequency. The information need not be kept secret.

### 9.4.2 Key Agreement Operation

A sequence of shared secret keys $K_1$, $K_2$, …, $K_t$ shall be generated by each party by performing the following or an equivalent sequence of steps:

1. Establish the valid set of DL or EC domain parameters with which the parties' two key pairs shall be associated.
2. Select a valid private key $s$ and a valid key pair ($u$, $v$) for the operation, associated with the parameters established in Step 1.
3. Obtain the other party's two purported public keys $w$ ¢and $v$ ¢for the operation, associated with the parameters established in Step 1.
4. *(Optional.)* If the selected secret value derivation primitive is DLSVDP-MQVC or ECSVDP-MQVC, then validate that $w$ ¢and $v$ ¢are elements in the appropriate group (i.e., in GF($q$) for DL or on the elliptic curve for EC—see Sections 6.2.4 and 7.2.4); otherwise validate that $w$ ¢and $v$ ¢are valid public keys. If any validation fails, output "invalid public key" and stop.
5. Compute a shared secret value $z$ from the selected private keys $s$ and $u$ and the other party's two public keys $w$ ¢and $v$ ¢with the selected secret value derivation primitive (see Section 9.4.1).
6. Convert the shared secret value $z$ to an octet string $Z$ using FE2OSP.
7. For each shared secret key to be agreed on:
   a. Establish or otherwise agree on key derivation parameters $P_i$ for the key.
   b. Derive a shared secret key $K_i$ from the octet string $Z$ and the key derivation parameters $P_i$ with the selected key derivation function (see Section 9.4.1).

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of domain parameters
— at least one valid private key $s$ for each set of domain parameters
— all valid key pairs ($u$, $v$) associated with the same set of domain parameters as $s$
— all valid public keys $w$ ¢and $v$ ¢associated with the same set of domain parameters as $s$; if key validation is performed or an -MQVC primitive is used, invalid public keys $w$ ¢and $v$ ¢that are appropriately handled by the implementation may also be included in the conformance region
— a range of key derivation parameters $P$

NOTE—These schemes, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.42 [ANS98b] (in the DL case) and ANSI X9.63 [ANS98f] (in the EC case).

# 10. Signature Schemes

The general model for signature schemes is given in Section 10.1. Two specific schemes and their allowable options are given in Sections 10.2 and10.3.

## 10.1 General Model

In a signature scheme, one party, the *signer*, generates a signature for a message with its own private key, and another party, the *verifier*, verifies the signature with the signer's corresponding public key. A signature scheme can provide assurance of the origin and integrity of a message, and can protect against impersonation of parties and modification of messages.

There are two types of signature schemes. In a *signature scheme with appendix*, the signer conveys both the message and the signature to the verifier, who verifies their consistency. In a *signature scheme with message recovery*, the signer conveys only the signature to the verifier, who verifies the signature and, if it is correct, recovers the message from it. This standard defines only signature schemes with appendix (see Annex C.3.4 and C.3.7).

A signature scheme consists of a signature generation operation and a signature verification operation, along with supporting key management. Domain parameter and key pair generation for the signature schemes are specified further in connection with the DL, EC and IF families (Sections 6, 7 and 8, respectively). Security considerations for signature schemes are given in Annex D.5.2.

A signature generation operation has the following form for all the schemes:

1. Select a valid private key (together with its set of domain parameters, if any) for the operation.
2. Apply certain cryptographic operations to the message and the private key to produce a signature.
3. Output the signature.

A signature verification operation in a signature scheme with appendix has the following form:

1. Obtain the signer's purported public key (together with its set of domain parameters, if any) for the operation. (Note 1.)
2. *(Optional.)* Validate the public key and its associated set of domain parameters, if any. If validation fails, output "invalid" and stop. (Note 2.)
3. Apply certain cryptographic operations to the message, the signature and the public key to verify the signature.
4. Output "valid" or "invalid" according to the result of step 3.

NOTES

1—*(Authentication of ownership.)* The process of obtaining the signer's public key (step 1 of verification) may involve authentication of ownership of the public key, as further described in D.3.1 and D.5.2.4. This may be achieved by verifying a certificate or by other means. The means by which the key (or the certificate containing it) is obtained may vary, and may include the signer sending the key to the verifier, or the verifier obtaining the key from a third party or from a local database.

2—*(Domain parameter and key validation.)* Since a signature verification primitive assumes that the domain parameters and the public key are valid, the result of the primitive is undefined otherwise. Consequently, it is recommended that the verifier validate the public key (and its associated set of domain parameters, if any) in step 3, unless the risk of operating on invalid domain parameters or keys is mitigated by other means, as discussed further in

D.3.2 and D.5.2.5. Examples of "other means" include validation within the supporting key management, or assignment of liability to a purported signer for all signatures that can be verified with the signer's public key, whether or not the public key is valid.

3—*(Error conditions.)* The signer's and verifier's steps may produce errors under certain conditions, such as the following:

— private key not found in signer's step 1
— message or private key not supported in signer's step 2
— public key not found in verifier's step 1
— public key not valid in verifier's step 2
— message, signature, or public key not supported in verifier's step 3

Such error conditions should be detected and handled appropriately by an implementation, but the specific methods for detecting and handling them is outside of the scope of this standard.

4—*(Repudiation.)* A dishonest signer may be interested in the ability to claim that something went wrong and the signature was not authentic in order to disclaim the liability for the signature. This is known as *repudiation*. See Annex D.5.2.3 for more on repudiation and ways to address it.

## 10.2 DL/ECSSA

DL/ECSSA is Discrete Logarithm and Elliptic Curve Signature Scheme with Appendix.

### 10.2.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme (the signer and the verifier):

— the signature and verification primitives, which shall be one of the following pair of primitives:
  — the pair DLSP-NR and DLVP-NR, the pair DLSP-DSA and DLVP-DSA, the pair ECSP-NR and ECVP-NR, or the pair ECSP-DSA and ECVP-DSA
— the message encoding method for signatures with appendix, which should be EMSA1 (Section 12.1.1), or a technique designated for use with DL/ECSSA (and the selected signature primitive) in an addendum to this standard.

The above information may remain the same for any number of executions of the signature scheme, or it may be changed at some frequency. The information need not be kept secret.

### 10.2.2 Signature Generation Operation

A signature ($c$, $d$) shall be generated by a signer from a message $M$ by the following or an equivalent sequence of steps:

1. Select a valid private key $s$ and its associate set of domain parameters for the operation.
2. If the selected signature primitive is DLSP-NR or ECSP-NR, then set the maximum length of the message representative to be $l$ = (length of $r$ in bits) – 1, where $r$ is the order of the base point in the DL or EC set of domain parameters.
3. If the selected signature primitive is DLSP-DSA or ECSP-DSA, then set the maximum length of the message representative to be $l$ = length of $r$ in bits.
4. Use the encoding operation of the selected message encoding method (see Section 10.2.1) to produce a message representative $f$ of maximum length $l$ from the message $M$ ($f$ will be a non-negative integer).

5. Apply the selected signature primitive (see Section 10.2.1) to the integer *f* and the private key *s* to generate the signature (*c*, *d*).
6. Output the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of domain parameters
— at least one valid private key *s* for each set of domain parameters
— a range of messages *M*

### 10.2.3 Signature Verification Operation

A signature (*c*, *d*) on a message *M* shall be verified by a verifier by the following or an equivalent sequence of steps:

1. Obtain the other party's purported public key *w* and its associated set of domain parameters for the operation.
2. *(Optional.)* Validate the public key *w* and its associated set of domain parameters. Output "invalid" and stop if validation fails.
3. If the selected verification primitive is DLVP-NR or ECVP-NR:
   a) Apply the selected verification primitive (see Section 10.2.1) to the signature (*c*, *d*) and the signer's public key to recover an integer *f*. If the output of the primitive is "invalid," output "invalid" and stop.
   b) Set the maximum length of the message representative to be *l* = (length of *r* in bits) – 1, where *r* is the order of the base point in the DL or EC set of domain parameters.
   c) Use the verification operation of the selected message encoding method (see Section 10.2.1) to verify that the integer *f* is a correct encoded representative of the message *M* according to the encoding method and maximum length *l*. If that is the case, output "valid." Else, output "invalid."
4. If the selected verification primitive is DLVP-DSA or ECVP-DSA:
   a) Set the maximum length of the message representative to be *l* = length of *r* in bits, where *r* is the order of the base point in the DL or EC set of domain parameters.
   b) Use the decoding operation of the selected message encoding method (see Section 10.2.1) to produce a message representative *f* of maximum length *l* from the message *M* (*f* will be a non-negative integer).
   c) Apply the selected verification primitive (see Section 10.2.1) to the integer *f*, the signature (*c*, *d*), and the signer's public key to determine whether they are consistent. If the output of the primitive is "invalid," output "invalid"; otherwise, output "valid."

**Conformance region recommendation.** A conformance region should include:
— at least one valid set of domain parameters
— at least one valid public key *w* for each set of domain parameters; if key validation is performed, invalid public keys *w* that are appropriately rejected by the implementation may also be included in the conformance region
— all messages *M* that can be input to the implementation
— all purported signatures (*c*, *d*) that can be input to the implementation; this should at least include all (*c*, *d*) such that *c* and *d* are in the range [1, *r* – 1] for -DSA or [0, *r* – 1] for -NR, where *r* is from the domain parameters of *w*

NOTES

1—Since message encoding methods for signatures with appendix are usually based on hash functions, the length of a message that can be signed by this scheme is usually unrestricted or constrained by a very large number.

2—Because of the way verification is performed when the verification primitive is DLVP-DSA or ECVP-DSA, the message encoding method has to be deterministic when one of these primitives is used. Unlike verification using DLVP-NR or ECVP-NR, where the message representative is an output of the verification primitive, for DLVP-DSA and ECVP-DSA the message representative is one of the inputs to the verification primitive.

3—These schemes, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.30 [ANS97a] (in the DL case), ANSI X9.62 [ANS98e] (in the EC case) and ISO/IEC 14888-3 [ISO98h].

## 10.3 IFSSA

IFSSA is Integer Factorization Signature Scheme with Appendix.

### 10.3.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme (the signer and the verifier):

— the type of key pair for the signer (RSA or RW)
— the signature and verification primitives, which shall be a pair from the following list:
  — if the signer's public key is an RSA public key, then the primitives shall be either the pair IFSP-RSA1 and IFVP-RSA1, or the pair IFSP-RSA2 and IFVP-RSA2
  — if the signer's public key is an RW public key, then the primitives shall be the pair IFSP-RW and IFVP-RW
— the message encoding method for signatures with appendix, which should be EMSA2 (Section 12.1.2), or a technique designated for use with IFSSA in an addendum to this standard (if the signature primitive is IFSP-RSA2 or IFSP-RW, the message encoding method shall always produce a message representative congruent to 12 modulo 16)

The above information may remain the same for any number of executions of the signature scheme, or it may be changed at some frequency. The information need not be kept secret.

### 10.3.2 Signature Generation Operation

A signature *s* shall be generated by a signer from a message *M* by the following or an equivalent sequence of steps:

1. Select a valid private key *K* for the operation.
2. Set the maximum length of the message representative to be $l$ = (length of $n$ in bits) – 1, where $n$ is the modulus in the IF private key.
3. Use the encoding operation of the selected message encoding method (see Section 10.3.1) to produce a message representative $f$ of maximum length $l$ from the message $M$ ($f$ will be a non-negative integer). If the selected signature primitive is IFSP-RSA2 or IFSP-RW, $f$ will be congruent to 12 modulo 16.
4. Apply the selected signature primitive (see Section 10.3.1) to the integer $f$ and the selected private key $K$ to generate the signature $s$.
5. Output the signature.

**Conformance region recommendation.** A conformance region should include:
— at least one valid private key *K*
— a range of messages *M*

### 10.3.3 Signature Verification Operation

A signature *s* on a message *M* shall be verified by a verifier by the following or an equivalent sequence of steps:

1. Obtain the signer's purported public key (*n*, *e*).
2. *(Optional.)* Validate the public key (*n*, *e*). Output "invalid" and stop if validation fails.
3. Apply the selected verification primitive (see Section 10.3.1) to the signature *s* and the signer's public key to recover an integer *f*. If the output of the primitive is "invalid," output "invalid" and stop.
4. Set the maximum length of the message representative to be *l* = (length of *n* in bits) – 1, where *n* is the modulus in the IF public key.
5. Verify that the integer *f* is a correct encoded representative of the message *M* using the verification operation of the selected message encoding method (see Section 10.3.1) and maximum length *l*. If that is the case, output "valid." Else, output "invalid."

**Conformance region recommendation.** A conformance region should include:

— at least one valid public key (*n*, *e*); if key validation is performed, invalid public keys (*n*, *e*) that are appropriately rejected by the implementation may also be included in the conformance region
— all messages *M* that can be input to the implementation
— all purported signatures *s* that can be input to the implementation; this should at least include all *s* in the range [0, *n* – 1] for IFVP-RSA1 or [0, (*n* – 1)/2] for IFVP-RSA2 and IFVP-RW

NOTES

1—The upper bound on the length in bits of the message representative is so that the message representative is less than the modulus *n*, as required by all signature primitives. Since message encoding methods for signatures with appendix are usually based on hash functions, the length of a message that can be signed by this scheme is usually unrestricted or constrained by a very large number.

2—This scheme, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.31 [ANS98a] and ISO/IEC 14888-3 [ISO98h].

# 11. Encryption Schemes

Discussion of encryption schemes in general is given in Section 11.1. A specific scheme and its allowable options are given in Section 11.2.

## 11.1 General Model

In an encryption scheme, one party, the *sender*, generates a ciphertext for a message with another party's public key. The other party, called the *recipient*, decrypts the ciphertext with the corresponding private key to obtain the original message. An encryption scheme can provide confidentiality of a message.

In addition to providing confidentiality of a message, an encryption scheme may provide for a secure association between the message and a string known as *encoding parameters*. The encoding parameters themselves are not encrypted by the encryption scheme, but are securely associated with the ciphertext by the sender. The recipient, during decryption, may also supply encoding parameters and verify whether or not the sender used the same encoding parameters during encryption. The encoding parameters may be an empty string. See Annex D.5.3.3 for more on encoding parameters.

An encryption scheme consists of an encryption operation and a decryption operation, along with supporting key management. Domain parameter and key pair generation for the encryption scheme below are specified further in connection with the IF family (Sections 8). Security considerations for encryption schemes are given in Annex D.5.3.

Since there is only one encryption scheme in the current version of the standard, the general form of an encryption scheme, similar to those for key agreement and signature schemes, is not presented here.

## 11.2 IFES

IFES is Integer Factorization Encryption Scheme.

### 11.2.1 Scheme Options

The following options shall be established or otherwise agreed upon between the parties to the scheme (the sender and the recipient):

— the message encoding method for encryption, which should be EME1 (Section 12.2.1), or a technique designated for use with IFES in an addendum to this standard

The above information may remain the same for any number of executions of the encryption scheme, or it may be changed at some frequency. The information need not be kept secret.

### 11.2.2 Encryption Operation

The ciphertext $g$ shall be generated by a sender from a message $M$ and encoding parameters $P$ by the following or an equivalent sequence of steps:

1. Obtain the recipient's purported public key $(n, e)$ for the operation. (Note 1.)
2. *(Optional.)* Validate the public key $(n, e)$. Stop if validation fails. (Note 2.)

3. Set the maximum length of the message representative to be $l$ = (length of $n$ in bits) – 1, where $n$ is the modulus in the IF private key.
4. Use the encoding operation of the selected message encoding method (see Section 11.2.1) to produce a message representative $f$ of maximum length $l$ from the message $M$ and the encoding parameters $P$ ($f$ will be a non-negative integer). If the message is too long, the encoding method may not be able to produce a representative of the selected length. In that case, output "error" and stop.
5. Compute the ciphertext $g$ from $f$ and the recipient's public key with the primitive IFEP-RSA.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA public key ($n$, $e$); if key validation is performed, invalid public keys ($n$, $e$) that are appropriately rejected by the implementation may also be included in the conformance region
— a range of messages $M$ and encoding parameters $P$ (where the range of encoding parameters may include only the empty string)

### 11.2.3 Decryption Operation

The plaintext $M$ shall be recovered from the ciphertext $g$ and encoding parameters $P$ by the following or an equivalent sequence of steps:

1. Select the private key $K$ for the operation.
2. Compute an integer $f$ from $g$ and the selected private key $K$ with primitive IFDP-RSA.
3. Set the maximum length of the message representative to be $l$ = (length of $n$ in bits) – 1, where $n$ is the modulus in the IF private key.
4. Decode the integer $f$ according to the decoding operation of the selected message encoding method (see Section 11.2.1), maximum length $l$ and the encoding parameters $P$ to produce the message $M$. If the decoding operation produces an error, output "invalid." Else output the message $M$ as the plaintext.

**Conformance region recommendation.** A conformance region should include:
— at least one valid RSA private key $K$
— all ciphertexts $g$ in the range [0, $n$ – 1], where $n$ is from the private key $K$
— a range of messages $M$ and encoding parameters $P$ (where the range of encoding parameters may include only the empty string)

NOTES

1—*(Authentication of ownership.)* The process of obtaining the recipient's public key (step 1 of encryption) may involve authentication of ownership of the public key, as further described in D.3.1 and D.5.3.4. This may be achieved by verifying a certificate or by other means. The means by which the key (or the certificate containing it) is obtained may vary, and may include the recipient sending the key to the sender, or the sender obtaining the key from a third party or from a local database.

2—*(Key validation.)* Since the encryption primitive assumes that the public key is valid, the result of the primitive is undefined if the public key is invalid. Consequently, it is recommended that the sender validate the public key (and its associated set of domain parameters, if any) in step 2, unless the risk of operating on invalid domain parameters or keys is mitigated by other means, as discussed further in D.3.2 and D.5.3.5. Examples of "other means" include validation within the supporting key management.

3—*(Error conditions.)* The sender's and recipient's steps may produce errors under certain conditions, such as the following:

— public key not found in sender's step 1

— public key not valid in sender's step 2
— message, encoding parameters, or public key not supported in sender's steps 4 and 5
— private key not found in recipient's step 1
— message, encoding parameters, or private key not supported in recipient's steps 2 and 4

Such error conditions should be detected and handled appropriately by an implementation, but the specific methods for detecting and handling them is outside of the scope of this standard.

4—*(Length of the message.)* The upper bound on the length in bits of the encoded message representative is so that, when converted to an integer, it is less than the modulus *n*, as required by the encryption primitive. Since the message representative has to, at the very least, contain the information contained in the message itself, the length of messages that can be encrypted by this scheme is limited by the modulus length and the selected message encoding method.

5—*(Compatibility.)* This scheme, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.44 [ANS98c].

# 12. Message Encoding Methods

This section describes message encoding methods used in this document as building blocks for schemes. Unlike cryptographic primitives and schemes, encoding methods do not use keys.

Each message encoding method consists of an encoding operation and of either a decoding or a verification operation. An encoding operation encodes the message to produce a non-negative integer, called a *message representative*. It takes the message, the maximum bit length $l$ of the output, and possibly other parameters as input, and produces the integer of bit length no more than $l$ (see Section 3.1 for the definition of bit length of an integer). A decoding operation gives back the original message given the message representative, the length $l$ and the same additional parameters as were passed to the encoding operation. A verification operation verifies whether the message representative is a valid encoding of a message given the message representative, the message, the length $l$ and the same parameters as were passed to the encoding operation.

Different message encoding methods are needed for different categories of schemes. The use of an inadequate encoding method may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the encoding methods contained in this section, or any encoding methods defined in an addendum to this standard.

## 12.1 Message Encoding Methods for Signatures with Appendix

This standard strongly recommends the use of one of the following message encoding methods for signature schemes with appendix.

### 12.1.1 EMSA1

EMSA1 is an encoding method for signatures with appendix based on a hash function. It is recommended for use with DLSSA and ECSSA (Section 10.2).

The method is parameterized by the following choice:

— a hash function *Hash* with output length *hLen* octets, which shall be SHA-1 (Section 14.1.1), or RIPEMD-160 (Section 14.1.2), or a technique designated for use with EMSA1 in an addendum to the standard

NOTE—EMSA1 cannot produce message representatives longer than the hash function output; for SHA-1 and RIPEMD-160, the maximum length is 160 bits.

#### 12.1.1.1 Encoding Operation

**Input:**

— a message, which is an octet string $M$ (depending on the hash function chosen, there may be a limitation on the length of $M$; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)

— the maximum bit length $l$ of the output

**Output:** a message representative, which is an integer $f \geq 0$ of bit length at most $\min(l, 8hLen)$; or "error"

The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1. If the length of *M* is greater than the length limitation ($2^{61} - 1$ octets for SHA-1 or RIPEMD-160), output "error" and stop.
2. Compute *Hash* (*M*) with the selected hash function to produce an octet string *H* of length *hLen* octets.
3. If $l < 8hLen$, convert *H* to a bit-string *HB* of length $8hLen$ bits with the primitive OS2BSP. Remove $8hLen - l$ rightmost bits of *HB*, and then convert the remaining *l* bits to an integer *f* using the primitive BS2IP.
4. Else convert *H* to an integer *f* using OS2IP.
5. Output *f* as the message representative.

#### 12.1.1.2 Verification Operation

**Input:**
— a message, which is an octet string *M* (depending on the hash function chosen, there may be a limitation on the length of *M*; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)
— the maximum bit length *l* of the message representative
— the message representative, which is an integer $f \geq 0$

**Output:** "valid" if *f* is a correct representative of *M*; "invalid" otherwise.

The validity indicator shall be computed by the following or an equivalent sequence of steps:

1. Use the Encoding Operation of EMSA1 to compute a message representative *g*, a non-negative integer. If the encoding operation outputs "error," output "invalid" and stop.
2. If $f = g$, output "valid." Else, output "invalid."

### 12.1.2 EMSA2

EMSA2 is an encoding method for signatures with appendix based on a hash function with some additional formatting, based on ANSI X9.31 [ANS98a]. It is recommended for use with IFSSA (Section 10.3).

The method is parameterized by the following choice:
— a hash function *Hash* with output length 20 octets, which shall be SHA-1 (Section 14.1.1), or RIPEMD-160 (Section 14.1.2), or a technique designated for use with EMSA2 in an addendum to the standard

NOTE—EMSA2 cannot produce message representatives shorter than 191 bits in length.

#### 12.1.2.1 Encoding Operation

**Input:**
— a message, which is an octet string *M* (depending on the hash function chosen, there may be a limitation on the length of *M*; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)
— the maximum bit length *l* of the output

**Output:** a message representative, which is an integer $f \geq 0$ of bit length at most *l*; or "error"

The message representative *f* shall be computed by the following or an equivalent sequence of steps:

1. If the length of $M$ is greater than the length limitation ($2^{61} - 1$ octets for SHA-1 or RIPEMD-160), or $l < 191$, output "error" and stop.
2. Compute *Hash* ($M$) with the selected hash function to produce an octet string $H$ of length 20 octets.
3. If $M$ is of length 0 (i.e., an empty string), let $P_1$ be a single octet with hexadecimal value 4b. Otherwise let $P_1$ be a single octet with hexadecimal value 6b.
4. Let $P_2$ be an octet string of length $\lfloor (l + 1) / 8 \rfloor - 24$ octets, each with hexadecimal value bb.
5. Let $P_3$ be a single octet with hexadecimal value 33 if the selected hash function is SHA-1 and let it be 31 if the selected hash function is RIPEMD-160.
6. Let $T = P_1 \| P_2 \| \text{ba} \| H \| P_3 \| \text{cc}$, where ba and cc are single octets represented in hexadecimal.
7. Convert $T$ to an integer $f$ using OS2IP.
8. Output $f$ as the message representative.

#### 12.1.2.2 Verification Operation

**Input:**

— a message, which is an octet string $M$ (depending on the hash function chosen, there may be a limitation on the length of $M$; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)
— the maximum bit length $l$ of the message representative
— the message representative, which is an integer $f \geq 0$

**Output:** "valid" if $f$ is a correct representative of $M$; "invalid"

The validity indicator shall be computed by the following or an equivalent sequence of steps:

1. If the length of $M$ is greater than the length limitation ($2^{61} - 1$ octets for SHA-1 or RIPEMD-160), or $l < 191$, output "invalid" and stop.
2. Convert $f$ to an octet string $T$ of length $\lfloor (l + 1) / 8 \rfloor$ octets using the primitive I2OSP.
3. If $M$ is of length 0 (i.e., an empty string), let $P_1$ be a single octet with hexadecimal value 4b. Otherwise let $P_1$ be a single octet with hexadecimal value 6b.
4. Verify that the leftmost octet of $T$ is equal to $P_1$, the next $\lfloor (l + 1) / 8 \rfloor - 24$ octets each have hexadecimal value bb, and the next octet after that has hexadecimal value ba. If not, output "invalid" and stop.
5. Verify that the second rightmost octet has hexadecimal value 33 if the selected hash function is SHA-1 and it is 31 if the selected hash function is RIPEMD-160. If not, output "invalid" and stop.
6. Verify that the rightmost octet has hexadecimal value cc. If not, output "invalid" and stop.
7. Remove the leftmost $\lfloor (l + 1) / 8 \rfloor - 22$ octets of $T$ and the rightmost 2 octets of $T$ to produce an octet string $H'$ of length 20 octets.
8. Apply the selected hash function to $M$ to produce an octet string $H$ of length 20 octets.
9. If $H = H'$ output "valid." Otherwise, output "invalid."

## 12.2 Message Encoding Methods for Encryption

This standard strongly recommends the use of the following message encoding method for encryption schemes.

### 12.2.1 EME1

EME1 is an encoding method for encryption based on the "enhanced OAEP (Optimal Asymmetric Encryption Padding)" ([BR95], [JM96]), a hash function (Section 14.1) and a mask generation function (Section 14.2). It is recommended for use with IFES (Section 11.2). It can produce message representatives of arbitrary length $l$; however, there are restrictions on the length of the message it can encode that depend on $l$.

The method is parameterized by the following choice:

— a hash function *Hash* with output length *hLen* octets, which shall be SHA-1 (Section 14.1.1), or RIPEMD-160 (Section 14.1.2), or a technique designated for use with EME1 in an addendum to the standard

— a mask generation function *G*, which shall be MGF1, or a technique designated for use with EME1 in an addendum to this standard

#### 12.2.1.1 Encoding Operation

**Input:**

— the maximum bit length $l$ of the output

— a message, which is an octet string $M$ of length $mLen \le \lfloor (l/8) - 2hLen - 1 \rfloor$ octets

— encoding parameters, which is an octet string $P$ ($P$ may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may be a limitation on the length of $P$; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)

**Output:** a message representative, which is an integer $f \ge 0$ of bit length at most $l$, or "error"

The message representative $f$ shall be computed by the following or an equivalent sequence of steps:

1. If the length of $P$ is greater than the length limitation ($2^{61} - 1$ octets for SHA-1 or RIPEMD-160), output "error" and stop.
2. Let $oLen = \lfloor l/8 \rfloor$ and $seedLen = hLen$. If $mLen > oLen - hLen - seedLen - 1$ then output "error" and stop.
3. Let $S$ be the octet string that consists of $oLen - mLen - hLen - seedLen - 1$ zero octets. Let $T$ be the octet string consisting of a single octet with hexadecimal value 01.
4. Let $M \Leftarrow S \,||\, T \,||\, M$.
5. Apply the hash function *Hash* to the parameters $P$ to produce an octet string *cHash* of length *hLen*.
6. Let $DB = cHash \,||\, M \,¢$ The length of $DB$ is $oLen - seedLen$ octets.
7. Generate a fresh, random octet string *seed* of length *seedLen* octets (for more on generation of random strings, see Annex D.6).
8. Apply the mask generation function $G$ to the string *seed* to produce an octet string *dbMask* of length $oLen - seedLen$ octets.
9. Let $maskedDB = DB \oplus dbMask$.
10. Apply the mask generation function $G$ to the string *maskedDB* to produce an octet string *seedMask* of length *seedLen* octets.
11. Let $maskedSeed = seed \oplus seedMask$.
12. Let $EM = maskedSeed \,||\, maskedDB$.
13. Convert $EM$ to an integer $f$ with the primitive OS2IP.
14. Output $f$ as the message representative.

#### 12.2.1.2 Decoding Operation

**Input:**

— the maximum bit length $l$ of the message representative

— the message representative, which is an integer $f \ge 0$

— encoding parameters, which is an octet string $P$ ($P$ may be an empty string); see Annex D.5.3.3 for possible uses (depending on the hash function chosen, there may a limitation on the length of $P$; for SHA-1 and RIPEMD-160, the maximum length is $2^{61} - 1$ octets)

**Output:** the message, which is an octet string $M$; or "error"

The message shall be decoded by the following or an equivalent sequence of steps:

1. If the length of *P* is greater than the length limitation ($2^{61} - 1$ octets for SHA-1 or RIPEMD-160), output "error" and stop.
2. Let *oLen* = $\lfloor l/8 \rfloor$ and *seedLen* = *hLen*. If *oLen* < *seedLen* + *hLen* + 1, output "error" and stop.
3. Convert $f$ to an octet string *EM* of length *oLen* with the primitive I2OSP. If the primitive outputs "error," output "error" and stop.
4. Let *maskedSeed* be the leftmost *seedLen* octets of *EM*, and *maskedDB* be the remaining octets.
5. Apply the mask generation function *G* to the string *maskedDB* to produce an octet string *seedMask* of length *seedLen* octets.
6. Let *seed* = *maskedSeed* ⊕ *seedMask*.
7. Apply the mask generation function *G* to the string *seed* to produce an octet string *dbMask* of length *oLen* – *seedLen* octets.
8. Let *DB* = *maskedDB* ⊕ *dbMask*.
9. Apply the hash function *Hash* to the parameters *P* to produce an octet string *cHash* of length *hLen*.
10. Compare the first *hLen* octets of *DB* to *cHash*. If they are not equal, output "error" and stop.
11. Let *M* ¢be all but the first *hLen* octets of *DB*.
12. Let *T* be the leftmost non-zero octet of *M* ¢ If *T* ≠ hexadecimal value 01, output "error" and stop.
13. Remove *T* and all octets to the left of it (which are all zero) from *M* ¢to produce an octet string *M*.
14. Output the message *M*.

# 13. Key Derivation Functions

This section describes key derivation functions used in this standard as building blocks for key agreement schemes. A key derivation function computes one or more shared secret keys from shared secret value(s) and other mutually-known parameters. The derived secret keys are usually used in symmetric cryptography.

The use of an inadequate key derivation function compromises the security of the key agreement scheme in which it is used. This standard strongly recommends the use of the key derivation functions contained in this section, or any key derivation functions defined in subsequent addenda to this standard.

## 13.1 KDF1

KDF1 is a more general version of a similar construction key derivation function construction in ANSI X9.42 ([ANS98b]). It is recommended for use with DL and EC key agreement schemes (Section 9).

The function is parameterized by the following choice:

— a hash function *Hash* with output length *hLen* octets, which shall be SHA-1 (Section 14.1.1), or RIPEMD-160 (Section 14.1.2), or a technique designated for use with KDF1 in an addendum to the standard

**Input:**

— a shared secret string, which is an octet string *Z* of length *zLen* octets

— key derivation parameters, which is an octet string *P* of length *pLen* octets (depending on the hash function chosen, there may be a limitation on *zLen* + *pLen*; for SHA-1 and RIPEMD-160, the maximum of *zLen* + *pLen* is $2^{61} - 1$)

**Output:** a shared secret key, which is an octet string *K* of length *hLen* octets; or "error"

The shared secret key *K* shall be computed by the following or an equivalent sequence of steps:

1. If *zLen* + *pLen* exceeds the length limitation ($2^{61} - 1$ for SHA-1 or RIPEMD-160), output "error" and stop.
2. Compute *hash*(*Z* || *P*) with the selected hash function to produce an octet string *K* of *hLen* octets.
3. Output *K* as the shared secret key.

NOTE—The interpretation and usage of the *hLen*-octet output *K* is beyond the scope of this standard. For example, the two parties may agree to use the first 128 bits (with parity adjusted) of *K* as a session key for two-key triple-DES [ANS98d].

# 14. Auxiliary Functions

## 14.1 Hash Functions

A hash function is a building block for many techniques described in Sections 12, 13, and 14. It takes a variable-length octet string as input and outputs a fixed-length octet string. The length of the input to a hash function is usually unrestricted or constrained by a very large number. The output depends solely on the input—a hash function is deterministic.

### 14.1.1 SHA-1

SHA-1 is defined in [FIP95].

**Input:** an octet string *M* of length *mLen* octets (*mLen* has to be less than $2^{61}$)

**Output:** a hash value, which is an octet string *H* of length 20 octets

The hash value shall be computed by the following or an equivalent sequence of steps:

1. Convert *M* to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2. Apply the Secure Hash Algorithm, revision 1, as described in FIPS 180-1 to *MB* to produce a bit string *HB* of length 160 bits.
3. Convert *HB* to an octet string *H* with the primitive BS2OSP. The length of *H* will be 20 octets.
4. Output *H* as the hash value.

### 14.1.2 RIPEMD-160

RIPEMD-160 is based on the work of [DBP96].

**Input:** an octet string *M* of length *mLen* octets (*mLen* has to be less than $2^{61}$)

**Output:** a hash value, which is an octet string *H* of length 20 octets

The hash value shall be computed by the following or an equivalent sequence of steps:

1. Apply the RIPEMD-160 hash algorithm as described in ISO/IEC 10118-3:1998, clause 7 to *M* to produce a bit string *HB* of length 160 bits.
2. Convert *HB* to an octet string *H* with the primitive BS2OSP. The length of *H* will be 20 octets.
3. Output *H* as the hash value.

## 14.2 Mask Generation Functions

This section describes a mask generation function used in this standard as a building block for EME1. A mask generation function takes as input an octet string and the desired length of the output, and outputs an octet string of that length. The lengths of both the input to and the output of a mask generation function are usually unrestricted or constrained by a very large number. The output depends solely on the input—a mask generation function is deterministic.

### 14.2.1 MGF1

MGF1 is a mask generation function based on a hash function, following the ideas of [BR95] and [BR96]. It is used with EME1 (Section 12.2.1).

The function is parameterized by the following choice:

— a hash function *Hash* with output length *hLen* octets, which shall be SHA-1 (Section 14.1.1), or RIPEMD-160 (Section 14.1.2), or a technique designated for use with MGF1 in an addendum to the standard

**Input:**

— an octet string *Z* of *zLen* octets (depending on the hash function chosen, there may be a limitation on *zLen*; for SHA-1 and RIPEMD-160, *zLen* shall be less than or equal to $2^{61} - 5$)

— the desired length of the output, which is a positive integer *oLen* (*oLen* shall be less than or equal to $hLen \times 2^{32}$)

**Output:** an octet string *mask* of length *oLen* octets; or "error"

The octet string *mask* shall be computed by the following or an equivalent sequence of steps:

1. If *zLen* exceeds the length limitation ($2^{61} - 5$ for SHA-1 or RIPEMD-160), or if $oLen > hLen \times 2^{32}$, output "error" and stop.
2. Let *M* be the empty string. Let $cThreshold = \lceil oLen / hLen \rceil$.
3. Let *counter* = 0.
   - 3.1 Convert *counter* to octet string *C* of length 4 octets using I2OSP.
   - 3.2 Compute *Hash*(*Z* || *C*) with the selected hash function to produce an octet string *H* of *hLen* octets.
   - 3.3 Let *M* = *M* || *H*.
   - 3.4 Increment *counter* by one. If *counter* < *cThreshold*, go to Step 3.1.
4. Output the leading *oLen* octets of *M* as the octet string *mask*.

NOTE—Since *counter* is only 32 bits, the maximum length of the output is at most $hLen \times 2^{32}$.