

IEEE Standard Specifications for Public-Key Cryptography

Sponsor

Microprocessor Standards Committee
of the
IEEE Computer Society

Approved 30 January 2000
IEEE-SA Standards Board

Approved 27 July 2000
American National Standards Institute

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 on a variety of techniques from which applications may select.

Keywords: digital signature, encryption, key agreement, public-key cryptography

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 25 August 2000. Printed in the United States of America.

Print: ISBN 0-7381-1956-3 SH94820
PDF: ISBN 0-7381-1957-1 SS94820

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

(This introduction is not part of IEEE Std 1363-2000, IEEE Standard 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 held 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 this 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.

Participants

The active participants in the IEEE Std 1363-2000 Working Group at the time this standard was completed and balloted were as follows:

Burt Kaliski, *Chair*

Terry S. Arnold, *Vice Chair*

Robert Schlafly, *Secretary*

Michael Markowitz, *Treasurer*

Yiquin Lisa Yin, *Technical Editor*

Benjamin Arazi
Ian Blake
Lily Chen
Louis Finkelstein
Walter Fumy
Donald B. Johnson
Shirley Kawamoto
Tetsutaro Kobayashi

David Kravitz
Pil Joong Lee
Franck Leprevost
Daniel Lieman
Alfred Menezes
Tatsuaki Okamoto
Minghua Qu
Anand Rajan
Leonid Reyzin
Allen Roginsky

Richard Schroeppel
Ari Singer
Jerry Solinas
Kazuo Takaragi
Ashok Vadekar
Scott Vanstone
William Whyte
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
Paulo S. L. M. Barreto
Mihir Bellare
Tom Berson
Simon Blake-Wilson
Eric Blossom
Uri Blumenthal
Mark Chen
Don Coppersmith
Richard Crandall
Wei Dai
Erik De Win
Jean-Francois Dhem
Whitfield Diffie
Carl Ellison
Amos Fiat
Robert Gallant
John Gilmore
Roger Golliver
Gary L. Graunke
Phillip Griffin
Louis Guillou
Stuart Haber

Shai Halevi
Shouichi Hirose
Robert Hofer
Dale Hopkins
David Hopwood
Russell Housley
Eric Hughes
David Jablon
Marc Joye
Aleksandar Jurisic
Charanjit Jutla
John Kennedy
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
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 this standard.

The following members of the balloting committee voted on this standard:

Carlisle M. Adams
Malcolm J. Airst
Christopher Allen
Terry S. Arnold
Glen Atkins
Simon Blake-Wilson
Lily Chen
John L. Cole
Erik De Win
Dante Del Corso
Stephen L. Diamond
Robert Gallant
Patrick S. Gonia
Julio Gonzalez-Sanz
Gary L. Graunke
Sandor V. Halasz
Jim D. Isaak
Donald B. Johnson
Burt Kaliski

Shirley Kawamoto
David Kravitz
Thomas M. Kurihara
Robert J. Lambert
Pil Joong Lee
Franck Leprevost
Thomas Luedeke
Michael J. Markowitz
Joseph R. Marshall
Serge Mister
Stig Frode Mjolsnes
Paul S. Montague
John E. Montague
Roy Oishi
Minghua Qu
Leonid Reyzin
David Rockwell
Roger Schlafly
Marius Seritan

W. Olin Sibert
Ari Singer
Keith Sollers
Michael Steinacker
Fred J. Strauss
Kazuo Takaragi
Joseph Tardo
Michael D. Teener
Ashok Vadekar
Paul Van Oorschot
Clarence M. Weaver, Jr
Michael J. Weiner
Harold M. Wilensky
Forrest D. Wright
Cheng-Wen Wu
Chung-Huang Yang
Yiqun Lisa Yin
Oren Yuen
Robert Zuccherato

When the IEEE-SA Standards Board approved this standard on 30 January 2000, it had the following membership:

Richard J. Holleman, *Chair*
Donald N. Heirman, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Dennis Bodson
Mark D. Bowman
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Ruben D. Garzon

James H. Gurney
Lowell G. Johnson
Robert J. Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
L. Bruce McClung
Daleep C. Mohla
Robert F. Munzner

Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Robert E. Hebner

Jennifer McClain Longman
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Purpose.....	1
1.3	Organization of the document.....	2
2.	References.....	2
3.	Definitions.....	3
4.	Types of cryptographic techniques	7
4.1	General model.....	7
4.2	Primitives	8
4.3	Schemes	8
4.4	Additional methods.....	9
4.5	Table summary.....	9
5.	Mathematical conventions	10
5.1	Mathematical notation	11
5.2	Bit strings and octet strings.....	12
5.3	Finite fields	13
5.4	Elliptic curves and points.....	14
5.5	Data type conversion.....	14
6.	Primitives based on the discrete logarithm problem.....	17
6.1	The DL setting	17
6.2	Primitives	19
7.	Primitives based on the elliptic curve discrete logarithm problem.....	27
7.1	The EC setting.....	27
7.2	Primitives	29
8.	Primitives based on the integer factorization problem	37
8.1	The IF setting	37
8.2	Primitives	39
9.	Key agreement schemes.....	46
9.1	General model.....	46
9.2	DL/ECKAS-DH1	47
9.3	DL/ECKAS-DH2.....	49
9.4	DL/ECKAS-MQV	50
10.	Signature schemes.....	51
10.1	General model.....	51

10.2 DL/ECSSA.....	53
10.3 IFSSA.....	55
11. Encryption schemes	56
11.1 General model.....	56
11.2 IFES	57
12. Message-encoding methods.....	58
12.1 Message-encoding methods for signatures with appendix	59
12.2 Message-encoding methods for encryption	61
13. Key derivation functions.....	63
13.1 KDF1.....	64
14. Auxiliary functions	64
14.1 Hash functions	64
14.2 Mask generation functions.....	65
Annex A (informative) Number-theoretic background.....	67
A.1 Integer and modular arithmetic: overview.....	67
A.2 Integer and modular arithmetic: algorithms.....	72
A.3 Binary finite fields: overview	77
A.4 Binary finite fields: algorithms	85
A.5 Polynomials over a finite field.....	91
A.6 General normal bases for binary fields	94
A.7 Basis conversion for binary fields.....	98
A.8 Bases for binary fields: tables and algorithms	104
A.9 Elliptic curves: overview	115
A.10 Elliptic curves: algorithms	121
A.11 Functions for elliptic curve parameter and key generation.....	131
A.12 Functions for elliptic curve parameter and key validation.....	134
A.13 Class group calculations	143
A.14 Complex multiplication	147
A.15 Primality tests and proofs.....	157
A.16 Generation and validation of parameters and keys	162
Annex B (normative) Conformance.....	170
B.1 General model.....	170
B.2 Conformance requirements	171
B.3 Examples.....	173
Annex C (informative) Rationale.....	177
C.1 General	177
C.2 Keys and domain parameters	178
C.3 Schemes	179

Annex D (informative) Security considerations.....	182
D.1 Introduction.....	182
D.2 General principles	182
D.3 Key management considerations	184
D.4 Family-specific considerations	188
D.5 Scheme-specific considerations	198
D.6 Random number generation.....	208
D.7 Implementation considerations	212
Annex E (informative) Formats	213
E.1 Overview.....	213
E.2 Representing basic data types as octet strings	213
E.3 Representing outputs of schemes as octet strings	216
Annex F (informative) Bibliography	217

IEEE Standard Specifications for Public-Key Cryptography

1. Overview

1.1 Scope

This standard covers specifications for 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 are also discussed. Classes of computers and communication systems are 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 an amendment to this standard. Its scope is similar to that of this standard, with the addition of identification schemes. (See the note in 1.2 for a description of the purpose of P1363a.)

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 standard to mandate any particular set of public-key techniques, or particular attributes of public-key techniques, such as key sizes. Rather, its 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 an amendment to this standard. Its purpose is similar to that of this standard; however, P1363a will focus on newer techniques, while this standard specifies relatively well-established techniques. (See the note in 1.1 for a description of the scope of P1363a.)

1.3 Organization of the document

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

1.3.1 Structure of the main document

- Clause 1 is an overview.
- Clause 2 provides references to other standards.
- Clause 3 defines relevant terms used throughout this standard.
- Clause 4 gives an overview of the types of cryptographic techniques that are described in this standard.
- Clause 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.
- Clause 6 through Clause 11 define three families of cryptographic techniques: techniques based on the discrete logarithm problem (DL), the elliptic curve discrete logarithm problem (EC), and the integer factorization problem (IF). Clause 6, Clause 7, and Clause 8 define the setting and primitives used in the DL, EC, and IF families, respectively. Clause 9, Clause 10, and Clause 11 define key agreement schemes, signature schemes, and encryption schemes, respectively.
- Clause 12 defines message-encoding methods for signature and encryption schemes described in Clause 10 and Clause 11.
- Clause 13 defines key derivation functions for key agreement schemes described in Clause 9.
- Clause 14 defines certain auxiliary functions supporting the techniques described in Clause 12 and Clause 13.

1.3.2 Structure of the annexes

The six annexes provide background and helpful information for the users of this standard, as follows:

- 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 the following publications.

FIPS PUB 180-1, Secure Hash Standard, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/National Institute of Standards and Technology, April 1995.¹

¹FIPS publications are available from the National Technical Information Service (NTIS), U. S. Dept. of Commerce, 5285 Port Royal Rd., Springfield, VA 22161 (<http://www.ntis.gov/>) and/or from the Federal Information Processing Standards Publications web site at <http://www.itl.nist.gov/fipspubs/>.

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, but not all, of the techniques in this standard.
- 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— A bibliography is provided in Annex F.

3. Definitions

For the purposes of this standard, the following terms and definitions apply. The IEEE Standard Dictionary of Electrical and Electronics Terms, Sixth Edition [B71]² should be referenced for terms not defined in this clause.

3.1 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.

NOTE—See D.3.2 for more information.

3.2 bit length: *See: length.*

3.3 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.

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

3.5 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.

NOTE—See B.1 for more information.

3.6 cryptographic families: Three families of techniques, which are 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).

3.7 decrypt: To produce plaintext from ciphertext. *Contrast: encrypt. See also: ciphertext, encryption, plaintext.*

3.8 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.*

3.9 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:*

²The numbers in brackets correspond to those of the bibliography in Annex F.

digital signature, public key, public-key cryptography, private key, signature scheme with appendix, signature scheme with message recovery.

3.10 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.**

3.11 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.**

3.12 encrypt: To produce ciphertext from plaintext. *Contrast:* **decrypt.** *See also:* **ciphertext, encryption, plaintext.**

3.13 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, such 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.**

3.14 family: *See:* **cryptographic families.**

3.15 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 discrete logarithm (DL) or elliptic curve (EC) scheme is always based on computations in a field.

NOTE—See Koblitz [B95] for a precise mathematical definition.

3.16 finite field: A field in which there are only a finite number of quantities. The discrete logarithm (DL) and elliptic curve (EC) schemes are always implemented over finite fields.

NOTE—See Clause 5 for a description of the particular finite fields used in this standard.

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

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

3.19 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.**

3.20 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 also:* **key agreement.**

NOTE—See D.5.1.3 for more information.

3.21 key derivation: The process of deriving a secret key from a secret value. *See also:* **secret key, secret value.**

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

3.23 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.**

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

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

3.26 least significant: *See:* **last bit, last octet.**

3.27 length: (A) The length of a bit string is the number of bits in the string. (B) The length of an octet string is the number of octets in the string. (C) The 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). (D) The 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.

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

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

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

3.31 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.**

3.32 most significant: *See:* **first bit, first octet.**

3.33 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.**

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

3.35 parameters: *See:* **domain parameters.**

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

3.37 private key: The private element of the public/private key pair. *See also:* **public/private key pair, valid key.**

3.38 public key: The public element of the public/private key pair. *See also:* **public/private key pair, valid key.**

3.39 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.**

3.40 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 be 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.**

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

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

3.43 root (of a polynomial): For a polynomial $f(x)$, a value r such that $f(r) = 0$.

3.44 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.**

3.45 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.**

3.46 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.**

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

3.48 signature: *See:* **digital signature.**

3.49 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.**

3.50 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.**

3.51 signature verification: The process of verifying a digital signature. *See also:* **digital signature, digital signature scheme.**

3.52 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.**

3.53 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 (e.g., it may be internally inconsistent) and thus not be valid. *See also:* **domain parameters, public/private key pair, valid key, valid key pair, validation.**

3.54 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 (e.g., 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.**

3.55 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 (e.g., 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.**

3.56 validation: *See:* **domain parameter validation, key validation.**

4. Types of cryptographic techniques

This clause 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 information on conformance.

4.1 General model

As stated in Clause 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 allows 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 by themselves, but they serve as building blocks for schemes.
- *Schemes:* A collection of related operations combining primitives and additional methods (see 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 4.2, 4.3, and 4.4, and specific techniques are defined in Clause 6 through Clause 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 information 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 of a 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 Clause 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 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 be performed properly, and, in some cases, validation also needs to be performed. Proper key management, which is outside the scope of this standard, 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 information 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
 - 1) Encoding methods for signatures with appendix (EMSA)
 - 2) Encoding method for encryption (EME)
- Key derivation functions (KDF), which are components of key agreement schemes
- Auxiliary functions, which are building blocks for other additional methods
 - 1) Mask generation function (MGF), which is used in EME
 - 2) 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 that 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

Table 1 gives a summary of all the schemes in this standard, together with the primitives and additional methods that are invoked within a scheme.

Table 1—Summary of techniques in the standard

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 OR DLSP-DSA and DLVP- DSA OR ECSP-NR and ECVN-NR OR ECSP-DSA and ECVN-DSA	EMSA1 (uses a hash function)
IFSSA	IFSP-RSA1 and IFVP-RSA1 OR IFSP-RSA2 and IFVP-RSA2 OR 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)
NOTE—Acronym definitions are as follows:		
<u>Families</u> DL: discrete logarithm EC: elliptic curve IF: integer factorization <u>Schemes</u> KAS: key agreement scheme SSA: signature scheme with appendix ES: encryption scheme <u>Primitives</u> SVDP: secret value derivation primitive SP: signature primitive VP: verification primitive EP: encryption primitive DP: decryption primitive		<u>Additional methods</u> KDF: key derivation function EMSA: encoding method for signatures with appendix EME: encoding method for encryption <u>Names of techniques</u> DH: Diffie–Hellman DHC: Diffie–Hellman with cofactor exponentiation/multiplication MQV: Menezes–Qu–Vanstone MQVC: Menezes–Qu–Vanstone with cofactor exponentiation/multiplication NR: Nyberg–Rueppel DSA: Digital signature algorithm RSA: Rivest–Shamir–Adleman RW: Rabin–Williams

5. Mathematical conventions

This clause 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 this standard:

0	Denotes the integer 0, the bit 0, or the additive identity (the element zero) of a finite field. See 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 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 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 5.4 for more on elliptic curves.
$\lceil x \rceil$	The smallest integer greater than or equal to the real number x . For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$, $\lceil -5 \rceil = -5$, and $\lceil -5.3 \rceil = -5$.
$\lfloor x \rfloor$	The largest integer less than or equal to the real number x . For example, $\lfloor 5 \rfloor = 5$, $\lfloor 5.3 \rfloor = 5$, $\lfloor -5 \rfloor = -5$, and $\lfloor -5.3 \rfloor = -6$.
$[a, b]$	The interval of integers between and including the integers a and b .
LCM (a, b)	For two positive integers a and b , denotes the <i>least common multiple</i> of a and b (i.e., the least positive integer that is divisible by both a and b). See 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 <i>greatest common divisor</i> of a and b (i.e., the largest positive integer that divides both a and b). See 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 , $0 \leq r < n$, 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 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 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 $ab \bmod n = 1$. Pronounced “the (multiplicative) inverse of a modulo n .” Also denoted by $1/a \bmod n$. See 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 <i>prime finite field</i> . See 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 <i>characteristic two finite field</i> . See 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 A.1.4 for a detailed description and A.2.3 for an algorithm to compute the Jacobi symbol.
O	The elliptic curve point at infinity. See 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 $e = 2.718\dots$) to the power a , where a is a real number.

NOTE—Throughout this main document, integers and field elements are denoted by lowercase letters, and octet strings and elliptic curve points are denoted by uppercase 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 lowercase letter in the DL and EC schemes (see 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 subclause 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 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 $0, 1, 2, \dots, p-1$.

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

5.3.2 Characteristic two finite fields

A *characteristic two finite field* (also known as a *binary 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 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 A.3 and A.4 for 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

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

shall be taken to represent the polynomial

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

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 traditional mathematical notation, the bits in this representation are indexed from right to left, as opposed to the bits in the normal basis representation (see 5.3.2.2), which are indexed from left to right (see also 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 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

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

shall be taken to represent the element

$$a_0\theta + a_1\theta^2 + a_2\theta^{2^2} + \dots + a_{m-1}\theta^{2^{m-1}}$$

where θ 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 traditional mathematical notation, the bits in this representation are indexed from left to right, as opposed to the bits in the polynomial basis representation (see 5.3.2.1), which are indexed from right to left. (See also 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 O . For 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 O) shall satisfy the following equation in $GF(q)$:

$$y_P^2 = x_P^3 + ax_P + b$$

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 O) shall satisfy the following equation in $GF(q)$:

$$y_P^2 + x_P y_P = x_P^3 + ax_P^2 + b$$

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

5.5 Data type conversion

This subclause 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 Annex E. Figure 1 shows the primitives presented in this subclause.

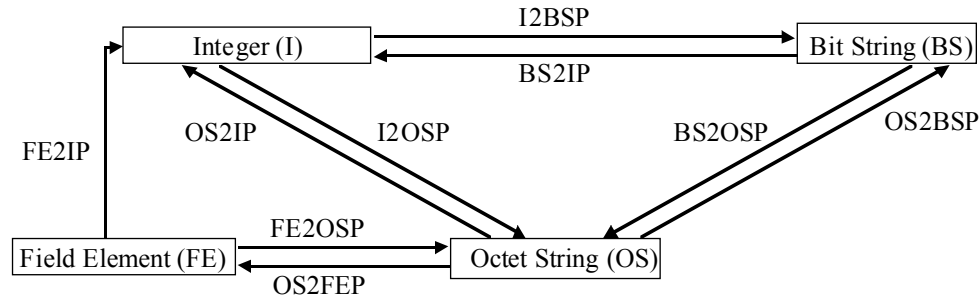


Figure 1 – Summary of data type conversion primitives

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

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

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 $1 \leq i \leq l$. The bit string shall be $b_1 b_2 \dots b_l$.

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 the 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 the 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 eight, and then breaks it up into octets. More precisely, a bit string $b_{l-1} b_{l-2} \dots b_0$ of length l shall be converted to an octet string $M_{d-1} M_{d-2} \dots M_0$ of length $d = \lceil l/8 \rceil$ as follows: for $0 \leq i < d-1$, let the octet $M_i = b_{8i+7} b_{8i+6} \dots b_{8i}$. The leftmost octet M_{d-1} shall have its leftmost $8d-l$ bits set to zero; its rightmost $8-(8d-l)$ bits shall be $b_{l-1} b_{l-2} \dots b_{8d-8}$.

The primitive that converts bit strings to octet strings is called the 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 the 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. OS2BSP 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

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

where $0 \leq x_i < 256$ (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 $0 \leq i \leq l-1$. The octet string shall be $M_{l-1} M_{l-2} \dots M_0$.

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 the 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 the 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 purposes of this standard, is represented by an integer if q is an odd prime (see 5.3.1) or by a bit string if q is a power of two (see 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 two, 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 the 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 two, 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 the 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 the Field Element to Integer Conversion Primitive, or FE2IP.

An 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 two, 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 clause 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 A.1.2, A.3.9, and Diffie and Hellman [B47].

6.1 The DL setting

6.1.1 Notation

The list below introduces the notation used in Clause 6, Clause 9, and Clause 10. It is meant for reference only; for complete definitions of the terms listed, refer to the appropriate text. Some other symbols are also used occasionally and 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' , and v' , respectively
w, v, w', v'	DL public keys, elements of $GF(q)$, corresponding to private keys s, u, s' , and 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
(c, d)	Signature, a pair of integers, computed by a signature primitive
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 , and v are used to denote the party's own keys, and the symbols s', u', w' , and 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 clause, operations on finite field elements and integers are 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 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 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 $\text{GCD}(k, r) = 1$ (i.e., r does not divide k ; see 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 discussed in D.3.2), and they may need to be validated (see below). The security issues related to domain parameter generation are discussed in D.3.1 and D.4.1. Suggested methods for generating DL domain parameters are discussed in 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 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 that is different from the one for which it was generated. A set of domain parameters may be shared by a number of key pairs (see 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.3.1 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 for 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 A.16.6. In some cases (when the DLSVDP-DHC or DLSVDP-MQVC primitive is applied), it may be necessary to verify only 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 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 subclause describes primitives used in the DL family. Before proceeding with this subclause, the reader should be familiar with the material in Clause 4.

As detailed in Clause 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. See Annex B for more on conforming with a primitive.

6.2.1 DLSVDP-DH

DLSVDP-DH is the Discrete Logarithm Secret Value Derivation Primitive, Diffie-Hellman version. It is based on the work in Diffie and Hellman [B47]. 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 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 the following:

- 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 D.5.1.6), which may occur valid. To prevent them, a key agreement scheme should validate the public key w' before executing this primitive (see also 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 the Discrete Logarithm Secret Value Derivation Primitive, Diffie-Hellman version with cofactor exponentiation. It is based on the work in Diffie and Hellman [B47], Kaliski [B88], and Law et. al. [B98]. 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 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 $GF(q)$; $\text{GCD}(k, r) = 1$

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 the following:

- 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 D.5.1.6). A key agreement scheme only needs to validate that w' is in $GF(q)$ before executing this primitive (see also 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 the Discrete Logarithm Secret Value Derivation Primitive, Menezes-Qu-Vanstone version. It is based on the work of Law et al. [B98]. 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 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 $z = \exp(v' \times \exp(w', t'), e)$.
7. If $z = 1$, output “error” and stop.
8. Output z as the shared secret value.

Conformance region recommendation: A conformance region should include the following:

- 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 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 6.2.4).

2—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. In this case, the primitive will output “error,” and will need to be rerun 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 the Discrete Logarithm Secret Value Derivation Primitive, Menezes-Qu-Vanstone version with cofactor exponentiation. It is based on the work in Kaliski [B88] and Law et al. [B98]. 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 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 $s, (u, v), w'$, 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'
- 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' and v' are in $GF(q)$; $\text{GCD}(k, r) = 1$.

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' into an integer i' using FE2IP.
4. Compute an integer $t' = i' \bmod 2^h$. Let $t' = t' + 2^h$.
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 $z = \exp(v' \times \exp(w', t'), ke)$.
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 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 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 rerun 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 the Discrete Logarithm Signature Primitive, Nyberg-Rueppel version. It is based on the work of Nyberg and Rueppel [B120]. 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 standard (see 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 $0 \leq f < r$

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

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

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 $c = i + f \bmod r$. 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 the following:

- 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.3.1, 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 the Discrete Logarithm Verification Primitive, Nyberg-Rueppel version. It is based on the work of Nyberg and Rueppel [B120]. 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 standard (see 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 $0 \leq f < r$; 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 the following:

- 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 include at least 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 the Discrete Logarithm Signature Primitive, DSA version. It is based on the work in Kravitz [B97]. 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 include at least 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.3.1, 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 the Discrete Logarithm Verification Primitive, DSA version. It is based on the work in Kravitz [B97]. 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 $c' = i \bmod r$.
6. If $c' = c$, then output “valid”; else, output “invalid.”

Conformance region recommendation: A conformance region should include:

- At least one valid set of 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 include at least 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 include at least 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 clause 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 A.9.3, A.9.4, Koblitz [B94], and Miller [B117].

7.1 The EC setting

7.1.1 Notation

The list below introduces the notation used in Clause 7, Clause 9, and Clause 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 and 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
(c, d)	Signature, a pair of integers, computed by a signature primitive
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

generally denoted by capital letters; for an elliptic curve point $P \neq O$, its x -coordinate and y -coordinate are denoted by x_P and y_P respectively: $P = (x_P, y_P)$.

3—Throughout this clause, operations on finite field elements, integers, and elliptic curve points are used. Care needs to be exercised to distinguish among these, as operations are denoted by the same symbols regardless of operands. (See 5.3 for more information on finite fields, and 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 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 $\text{GCD}(k, r) = 1$ (i.e., r does not divide k ; see 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 discussed in D.3.2), and they may need to be validated (see below). The security issues related to domain parameter generation are discussed in D.3.1 and D.4.2. A suggested method for generating EC domain parameters is contained in A.16.7 (see also 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 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 $W = sG$. (Note that $W \neq O$, 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 that is different from the one for which it was generated. A set of domain parameters may be shared by a number of key pairs (see 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.3.1 and D.4.2. A suggested method for generating EC key pairs is contained in A.16.9.

As is 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 for 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 A.16.10. In some cases (when the ECSVDP-DHC or ECSVDP-MQVC primitive is applied), it may be necessary to verify only 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 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 subclause describes primitives used in the EC family. Before proceeding with this subclause, the reader should be familiar with the material of Clause 4.

As detailed in Clause 4 and Annex B, an implementation of a primitive may make certain assumptions about its inputs, as listed with the specification for 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 $W = sG$ 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 the Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version. It is based on the work in Diffie and Hellman [B47], Koblitz [B94], and Miller [B117]. 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 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 $P = O$, 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 the following:

- 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 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 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 the Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version with cofactor multiplication. It is based on the work of Diffie and Hellman [B47], Kaliski [B88], Koblitz [B94], Law et al. [B98], and Miller [B117]. 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 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 $GF(q)$; $\text{GCD}(k, r) = 1$.

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 $P = ktW'$.
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 the following:

- 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 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 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 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 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 the Elliptic Curve Secret Value Derivation Primitive, Menezes-Qu-Vanstone version. It is based on the work of Law et al. [B98]. 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 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 $P = e(V' + t'W')$.
7. If $P = O$, 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 the following:

- 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 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 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 O . In this case, the primitive will output “error,” and will need to be rerun 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 the Elliptic Curve Secret Value Derivation Primitive, Menezes-Qu-Vanstone version with cofactor multiplication. It is based on the work of Kaliski [B88] and Law et al. [B98]. 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 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 $GF(q)$; $\text{GCD}(k, r) = 1$.

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' into an integer i' using FE2IP.
4. Compute an integer $t' = i' \bmod 2^h$. Let $t' = t' + 2^h$.
5. If compatibility with ECSVDP-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 an elliptic curve point $P = ke(V' + t'W')$.
7. If $P = O$, 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 the following:

- 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' and V' 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' and V' are not valid (see 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 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 A.12.3.

3—When the public key W' 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 O , and the primitive will output “invalid public key.” In this case, the primitive will need to be rerun 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 the Elliptic Curve Signature Primitive, Nyberg-Rueppel version. It is based on the work of Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. 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 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 $0 \leq f < r$

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

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

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 $V = (x_V, y_V)$ ($V \neq O$ 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 $c = i + f \bmod r$. 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 the following:

- 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.3.1, 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 the Elliptic Curve Verification Primitive, Nyberg-Rueppel version. It is based on the work of Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. 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 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 $0 \leq f < r$; 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 $P = dG + cW$. If $P = O$, 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 the following:

- 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 include at least 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 the Elliptic Curve Signature Primitive, DSA version. It is based on the work of Koblitz [B94], Kravitz [B97], and Miller [B117]. 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 $V = (x_V, y_V)$ ($V \neq O$ 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 the following:

- 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 include at least 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.3.1, 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 the Elliptic Curve Verification Primitive, DSA version. It is based on the work of Koblitz [B94], Kravitz [B97], and Miller [B117]. 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 $P = h_1 G + h_2 W$. If $P = O$, 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 the following:

- 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 include at least 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 include at least 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 clause 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 A.1.3, A.1.4, and Rivest, Shamir, and Adleman [B129].

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 Rivest, Shamir, Adleman [B129]); the other is known as RW, for “Rabin-Williams” (see Rabin [B128] and Williams [B149]).

8.1 The IF setting

8.1.1 Notation

The list below introduces the notation used in Clause 8, Clause 10, and Clause 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 and 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 work only with a particular fixed public exponent (see Note 5 in D.4.3.4).

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 ($3 \leq e < n$), which is an integer relatively prime to $(p - 1)$ and $(q - 1)$. The corresponding RSA *private exponent* is an integer d ($1 \leq d < n$) 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 ($2 \leq e < n$), 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 ($1 \leq d < n$) such that

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

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.3.1 and D.4.3. A suggested method for generating RSA key pairs is contained in A.16.11. A suggested method for generating RW key pairs is contained in 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 subclause describes primitives used in the IF family. Before proceeding with this subclause, the reader should be familiar with the material of Clause 4.

As detailed in Clause 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, $3 \leq e < n$, and e is relatively prime to $(p-1)(q-1)$]. The behavior of an implementation is not specified in the case where 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 subclause 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 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 $0 \leq i < n$ (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; $0 \leq i < n$.

Output: An integer j such that $0 \leq j < n$, 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 the Encryption Primitive. It is based on the work of Rivest, Shamir, and Adleman [B129]. 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 $0 \leq f < n$

Assumptions: Public key (n, e) is valid; $0 \leq f < n$.

Output: The encrypted message representative, which is an integer g such that $0 \leq g < n$

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 the following:

- 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 the RSA Decryption Primitive. It is based on the work of Rivest, Shamir, and Adleman [B129]. 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 $0 \leq g < n$

Assumptions: Private key K is valid; $0 \leq g < n$.

Output: The message representative, which is an integer f such that $0 \leq f < n$

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 the following:

- 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 the RSA Signature Primitive, version 1. It is based on the work of Rivest, Shamir, and Adleman [B129]. 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 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 $0 \leq f < n$

Assumptions: Private key K is valid; $0 \leq f < n$.

Output: The signature, which is an integer s such that $0 \leq s < n$

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 the following:

- 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 the RSA Verification Primitive, version 1. It is based on the work of Rivest, Shamir, and Adleman [B129]. 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 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.