

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1363.3™-2013

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

Sponsor

Microprocessor Standards Committee
of the
IEEE Computer Society

Approved 22 August 2013

IEEE-SA Standards Board

Abstract: Common identity-based public-key cryptographic techniques that use pairings, including mathematical primitives for secret value (key) derivation, public-key encryption, and digital signatures, as well as cryptographic schemes based on those primitives are specified in this standard. Also, related cryptographic parameters, public keys and private keys, are specified. The purpose of this standard is to provide a reference for specifications of a variety of techniques from which applications may select.

Keywords: encryption, identity-based encryption, IEEE 1363.3™, pairing-based cryptography, pairing-based encryption, public-key cryptography

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

Copyright © 2013 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 15 November 2013. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-0-7381-8649-8 STD98390
Print: ISBN 978-0-7381-8650-4 STDPD98390

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.

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.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

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.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive 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 comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, 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.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://ieeexplore.ieee.org/xpl/standards.jsp> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Patents

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 by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was completed, the P1363 Working Group had the following membership:

William Whyte, *Chair*
Don Johnson, *Vice Chair*

Kendall Ananyi
Matt Ball
Xavier Boyen
Mike Brenner
Daniel Brown
Mark Chimley
Andy Dancer
Mike Geipel

David Jablon
Satoru Kanno
Tetsutaro Kobayashi
David Kravitz
Phil MacKenzie
Michael Markowitz
Luther Martin
Marc Provencher
Jim Randall

Roger Schlafly
Mike Scott
Hovav Shacham
Ari Singer
Terence Spies
Yongge Wang
Tom Wu
Go Yamamoto

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Ed Addario
Mike Brenner
Keith Chow
Andy Dancer
James Davis
Thomas Dineen
Andrew Fieldsend
Randall C. Groves
Werner Hoelzl
Atsushi Ito
Mark Jaeger

Piotr Karocki
Thomas Kurihara
Susan Land
William Lumpkins
Greg Luri
Michael S. Newman
Nick S.A. Nikjoo
Randall Safier
Bartien Sayogo
Gil Shultz

Kapil Sood
Thomas Starai
Rene Struik
Walter Struppler
Joseph Tardo
Srinivasa Vemuru
John Vergis
Karl Weber
William Whyte
Oren Yuen
Janusz Zalewski

When the IEEE-SA Standards Board approved this standard on 22 August 2013, it had the following membership:

Richard H. Hulett, *Chair*
John Kulick, *Vice Chair*
Robert Grow, *Past Chair*
Judith Gorman, *Secretary*

Satish Aggarwal
Masayuki Ariyoshi
Peter Balma
William Bartley
Ted Burse
Clint Chaplin
Wael Diab
Jean-Philippe Faure

Alexander Gelman
Paul Houzé
Jim Hughes
Young Kyun Kim
Joseph L. Koepfinger*
David J. Law
Thomas Lee
Hung Ling

Oleg Logvinov
Ted Olsen
Gary Robinson
Jon Walter Rosdahl
Mike Seavey
Yatin Trivedi
Phil Winston
Yu Yuan

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Don Messina
IEEE Standards Program Manager, Document Development

Joan Woolery
IEEE Client Services Manager, Professional Services

Introduction

This introduction is not part of IEEE Std 1363.3-2013, IEEE Standard for Identity-Based Cryptographic Techniques using Pairings.

This standard describes eight identity-based cryptographic schemes that use pairings in their implementation. The schemes include approaches to encryption, digital signatures, signcryption, and key exchanges. These schemes may be used to encrypt both stored data as well as data in transit. An underlying mathematical operation called a “pairing” is a common element of these schemes, and the standard describes algorithms for calculating pairings and gives parameters suitable for implementing the specified schemes at industry-standard security levels (NIST SP 800-57 [B125]).^a

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

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 Organization of the document.....	2
2. Normative references.....	3
3. Definitions	3
4. Types of cryptographic techniques.....	7
4.1 General model.....	7
4.2 Primitives.....	8
4.3 Schemes.....	9
4.4 Table summary	10
5. Mathematical conventions.....	11
5.1 Mathematical notation	11
5.2 Bit strings and octet strings.....	13
5.3 Finite fields.....	13
5.4 Elliptic curves and points.....	16
5.5 Pairings	16
5.6 Data type conversion	16
6. Hashing primitives.....	23
6.1 Hashing to an integer.....	23
6.2 Hashing to a string.....	24
6.3 Hashing to a point in a subgroup	25
6.4 Hashing to an element of a finite field.....	29
7. Pairing-based primitives.....	30
7.1 General	30
7.2 SK primitives.....	30
7.3 BB ₁ primitives	33
7.4 BF primitives	36
7.5 SCC key agreement primitives	38
8. Identity-based encryption schemes.....	39
8.1 SK KEM scheme	40
8.2 BB ₁ KEM scheme.....	42
8.3 BB ₁ IBE scheme	44
8.4 BF IBE scheme	46
9. Identity-based signature schemes	48
9.1 BLMQ signature scheme	48
10. Identity-based signcryption schemes.....	50
10.1 BLMQ signcryption scheme.....	50
11. Identity-based key agreement schemes	53
11.1 Wang key agreement scheme	54
11.2 SCC key agreement scheme	57

Annex A (informative) Number-theoretic background	59
Annex B (normative) Conformance	121
Annex C (informative) Rationale	126
Annex D (informative) Security considerations	127
Annex E (informative) Formats.....	128
Annex F (informative) Bibliography	131

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

1. Overview

1.1 Scope

This document specifies identity-based cryptographic schemes based on the bilinear mappings over elliptic curves known as pairings. Specific techniques include algorithms to compute the pairings and specification of recommended elliptic curves and curve parameters over which the pairings are defined. The class of computer and communications systems is not restricted.

1.2 Purpose

The proliferation of electronic communication and the Internet brings with it the need for privacy and data protection. Public-key cryptography offers fundamental technology addressing this need. Many alternative public-key techniques have been proposed, each with its own benefits. IEEE Std 1363TM-2000¹ and IEEE Std 1363aTM-2004 have produced a comprehensive reference defining a range of common public-key techniques covering key agreement, public-key encryption, and digital signatures from several families, namely the discrete logarithm, integer factorization, and elliptic curve families. This document will specify identity-based cryptographic techniques based on pairings. These offer advantages over classic public-key

¹ Information on references can be found in Clause 2.

techniques specified in IEEE Std 1363-2000. Examples are the lack of a requirement to exchange or look up public keys of a recipient and the simplified use of short-lived keys. The class of computer and communications systems is not restricted.

1.3 Organization of the document

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

1.3.1 Structure of the main document

The structure of the main document is as follows:

- Clause 1 is a general overview.
- Clause 2 provides references to other standards and publications.
- Clause 3 defines some terms used throughout this standard.
- Clause 4 gives an overview of the types of cryptographic techniques that are defined in this standard.
- Clause 5 defines 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 defines certain cryptographic hashing primitives that are used to build the complete schemes described in subsequent clauses.
- Clause 7 defines certain pairing-based primitives that are used to build the identity-based schemes that are defined in the subsequent clauses.
- Clause 8 defines four pairing-based, identity-based encryption schemes.
- Clause 9 defines a pairing-based, identity-based signature scheme.
- Clause 10 defines a pairing-based, identity-based signcryption scheme.
- Clause 11 defines two pairing-based, identity-based key agreement schemes.

1.3.2 Structure of the annexes

The annexes provide background and helpful information for the users of the standard and consist of the following material:

- 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. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1363™-2000, IEEE Standard Specifications for Public-Key Cryptography.^{2,3}

IEEE Std 1363a™-2004, IEEE Standard Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques.

3. Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary: Glossary of Terms and Definitions*⁴ should be consulted for terms not defined in this clause.

authentication of ownership: The assurance that a given, identified party intends to be associated with a given public key. This may also include assurance that the party possesses the corresponding private key (see D.3.2 in IEEE Std 1363-2000 for more information).

bit length: *See: length.*

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

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

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

cryptographic family: Three families of techniques 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).

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

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

² IEEE publications are available from The Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

³ The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

⁴ *IEEE Standards Dictionary: Glossary of Terms and Definitions* is available at <http://shop.ieee.org>.

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

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

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

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

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

family: *See:* **cryptographic family.**

field: A setting in which the usual mathematical operations (addition, subtraction, multiplication, and division by nonzero quantities) are possible and obey the usual rules (such as the commutative, associative, and distributive laws). A discrete logarithm (DL) or elliptic curve (EC) scheme is always based on computations in a field. See Koblitz [B98]⁵ for a precise mathematical definition.

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

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

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

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

key confirmation: The assurance provided to each party participating in a key agreement protocol that the other party is capable of computing the agreed-upon key and that it is the same for both parties (see D.5.1.3 in IEEE Std 1363-2000 for more information). *See also:* **key agreement.**

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

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

⁵ The numbers in brackets correspond to those of the bibliography in Annex F.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Types of cryptographic techniques

This 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 on conformance.

4.1 General model

This document provides a reference for specifications of a variety of pairing-based public-key cryptographic techniques from which applications may select, and it defines these techniques in a framework that allows the selection of techniques appropriate for particular applications. Pairing-based cryptography enables either more compact versions of traditional cryptographic methods, such as short signature schemes, or key management techniques that can map an application-chosen identity string to a public key. In some circumstances, these identity-based techniques can yield more efficient or easier-to-use protocols.

The framework for pairing-based cryptographic techniques is similar to that defined in IEEE Std 1363a-2004, in that number-theoretic hard problems are used as the basis for cryptographic schemes that are incorporated into protocols. Pairing-based cryptography uses a different, but related, set of problems that are presumed to be computationally infeasible at appropriate sizes. These problems, typically variants of the bilinear Diffie-Hellman (BDH) problem, are close relatives of the Diffie-Hellman problem cited in IEEE Std 1363-2000.

Different types of cryptographic techniques can be viewed abstractly according to the following three-level general model:

- Primitives—Basic mathematical operations that are based on number-theoretic hard problems. Primitives are not meant to achieve security just by themselves, but the primitives serve as building blocks for schemes.
- Components—Cryptographic operations comprising primitives and other mathematical operations that in turn comprise schemes.
- Schemes—A collection of related operations combining primitives and components. Schemes can provide complexity-theoretic security, which is enhanced when the schemes 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).

A general framework of primitives is provided in Clause 5 through Clause 7; specific schemes are defined in Clause 8 through Clause 11. This standard, however, does not define protocols. These are 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:

- Cryptographically hashing an octet string to either an integer or a point on an elliptic curve
- Generation of an identity-based private key
- Verification of an identity-based private key
- Encryption of a plaintext message
- Decryption of a ciphertext message
- Generation of an identity-based signature
- Verification of an identity-based signature
- Reencryption of a ciphertext into a different ciphertext
- Derivation of an identity-based public key
- Derivation of a shared secret from public and private keys
- Encapsulation of a random session key
- Decapsulation of an encrypted session key
- Setup of system parameters needed for the operation of an identity-based cryptographic scheme

Primitives in this standard are presented as mathematical operations and are useful only as building blocks for the full schemes that follow.

Primitives assume that their inputs satisfy certain input constraints, as listed with the specification of each primitive. An implementation of a primitive is unconstrained on an input not satisfying the input constraints, 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
- Input constraints about the input made in the description of the operation performed by the primitive
- Output from the primitive
- Operation performed by the primitive, expressed as a series of steps
- Conformance region recommendations describing the minimum recommended set of inputs for which an implementation should operate in conformance with the primitive (see Annex B for more on conformance)

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

4.3 Schemes

The following types of schemes are defined in this standard:

- Identity-based encryption and key encapsulation
- Identity-based signatures
- Identity-based signcryption
- Identity-based key agreement
- Identity-based proxy reencryption

The goal of these schemes is to allow encrypted communication between multiple parties assuming the presence of one or more trusted key servers. These schemes allow the sending party to transform a string representing the identity of the receiving party a set of key server parameters into a public key. This public key can then be used to encrypt or derive a symmetric key, typically used with a symmetric encryption algorithm to encrypt a message to the second party. The receiving party shall then request a private key for that identity string from a key server. The key server uses a derivation operation to calculate the receiver's private key, which is communicated back to the receiver. Subsequent communications to the receiver that use the same identity string can be decrypted with a stored version of this key.

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. Although outside the scope of this standard, proper key management is essential for security. It is addressed in more detail in Annex D.

The specification of a scheme consists of the following information:

- Scheme options, such as choices for primitives and additional methods
- One or more operations, depending on the scheme, expressed as a series of steps

- Conformance region recommendations for implementations conforming with the scheme (see Annex B for more on conformance)

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

4.4 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 schemes and their components

Scheme name	Components	Primitives
SK KEM	SK-KEM-S SK-KEM-EX SK-KEM-EN SK-KEM-DE	P-SK-G P-SK-V P-SK-E P-SK-D IHF1 SHF1
BB ₁ KEM	BB1-KEM-S BB1-KEM-EX BB1-KEM-EN BB1-KEM-DE	P-BB1-G P-BB1-V P-BB1-E P-BB1-D IHF1 SHF1
BB ₁ IBE	BB1-IBE-S BB1-IBE-EX BB1-IBE-EN BB1-IBE-DE	P-BB1-G P-BB1-V P-BB1-E P-BB1-D IHF1 SHF1
BF IBE	BF-IBE-S BF-IBE-EX BF-IBE-EN BF-IBE-DE	P-BF-G P-BF-V P-BF-E P-BF-D IHF1 SHF1 PHF-SS PHF-GFP PHF-GF2
BLMQ signature	BLMQ-SIG-S BLMQ-SIG-EX BLMQ-SIG-SI BLMQ-SIG-VE	P-BLMQ-G P-BLMQ-GV
BLMQ signcryption	BLMQ-SC-S BLMQ-SC-EX BLMQ-SC-SE BLMQ-SC-DV	P-BLMQ-G P-BLMQ-GV
Wang key agreement	WKA-KA-D1 WKA-KA-D2 WKA-KA-V WKA-KA-D3 WKA-KA-G	P-BF-D P-BF-G PHF-SS PHF-GFP PHF-GF2
SCC key agreement	SCC-KA-G	P-BF-G P-BF-V P-SCC-D1

5. Mathematical conventions

This clause describes certain mathematical conventions used in the standard, including notation, terminology, 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 in 5.1.

5.1 Mathematical notation

The following mathematical notation is used throughout the document.

0 denotes the integer 0, the bit 0, or the additive identity (the element zero) of a finite field. See 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.

$a \times b$ denotes 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.

$a \times P$ denotes 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.

$\lceil x \rceil$ denotes the smallest integer greater than or equal to the real number x . For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.

$\lfloor x \rfloor$ denotes the largest integer less than or equal to the real number x . For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.

$[a, b]$ denotes the interval of integers between and including the integers a and b .

$LCM(a, b)$ denotes the least common multiple of a and b for two positive integers 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.

$GCD(a, b)$ denotes the greatest common divisor of a and b for two positive integers 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.

$X \oplus Y$ denotes the bitwise exclusive-or (XOR) of two bit strings or two octet strings X and Y of the same length.

$X \parallel Y$ denotes the ordered concatenation of two strings X and Y . X and Y are either both bit strings or both octet strings.

$\lg x$ denotes the logarithmic function of a positive number x to the base 2.

$\log_{256} x$ denotes the logarithmic function of a positive number x to the base 256.

$a \bmod n$ denotes 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.

$a \equiv b \pmod{n}$ denotes that $n \mid (a - b)$, so that the integers a and b have the same remainder when divided by the positive integer n . It is 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.

$a \not\equiv b \pmod{n}$ denotes that $n \nmid (a - b)$, so that the integers a and b have different remainders when divided by the positive integer n . It is pronounced “ a is not congruent to b modulo n .” This is equivalent to $(a \bmod n) \neq (b \bmod n)$.

$a^{-1} \bmod n$ denotes a positive integer $b < n$, if it exists, such that $(ab) \bmod n = 1$. This is pronounced “the (multiplicative) inverse of a modulo n ” and is also denoted by $\frac{1}{a} \bmod n$. See Annex A for a more detailed description and an algorithm for computing it.

$\frac{a}{b} \bmod n$ denotes an integer a multiplied by the inverse of the integer b , with the computations performed modulo n . It is equivalent to $ab^{-1} \bmod n$.

$GF(p)$ denotes the finite field of p elements, represented as the integers modulo p , where p is a prime number. This is also known as a prime finite field. See 5.3.1 for more information.

$GF(p^m)$ denotes the finite field containing p^m elements for some integer $m > 1$, where p is a prime number. If $n \mid m$, then this is also known as an extension field of the field $GF(p^n)$.

$GF(q)$ denotes the finite field containing q elements. In the context of this document, q will be a power of a prime.

$E / GF(q)$ denotes an elliptic curve defined over the field $GF(q)$.

$E' / GF(q)$ denotes the order twist of order t of the elliptic curve $E / GF(q)$.

$E(GF(q))$ denotes the additive group of points on the elliptic curve $E / GF(q)$.

$E(GF(q))[n]$ denotes the subgroup of $E(GF(q))$ consisting of all points of order n .

$\#E(GF(q))$ denotes the order of the group $E(GF(q))$ or the number of points on an elliptic curve defined over the field $GF(q)$. This may be abbreviated to $\#E$ when the context is clear.

$e(P, Q)$ denotes a pairing, or an efficiently computable, nondegenerate bilinear mapping. For an elliptic curve $E / GF(q)$ and an integer $n \mid \#E(GF(q))$, such a pairing takes as parameters elliptic curve points $P \in E(GF(q))[n]$ and $Q \in E(GF(q^k))$, and evaluates as an element in the multiplicative group $GF(q^k)^*$. The integer k is known as the embedding degree and is the smallest integer k such that $n \mid q^k - 1$.

$e_2(P, Q)$ denotes a pairing, the output of which is compressed by a factor of 2 and represented as an element in $GF(p^{k/2})$.

$e_3(P, Q)$ denotes a pairing, the output of which is compressed by a factor of 3 and represented as an element in $GF(p^{k/3})$.

$\phi(P)$ denotes a distortion map or a nonrational endomorphism on an elliptic curve group $E(GF(q^k))$. Such a mapping maps a point $P \in E(GF(q))$ to a point $\phi(P) \in E(GF(q^k))$ such that P and $\phi(P)$ are linearly independent.

$\phi_d(P)$ denotes a mapping from $E^d(GF(q))$ to $E(GF(q^d))$. If k is the embedding degree of E , then let ϵ be the degree of the maximal twist of E . Then $\epsilon \mid k$, and we define the degree of the field over which we consider the twist to be $d = k/\epsilon$.

(x / n) denotes a Jacobi symbol. See A.1.3.2 for a detailed description and A.4.3 for an algorithm to compute Jacobi symbols.

$\mathcal{O}$ denotes the point at infinity on an elliptic curve. See 5.4 for more information.

$\exp(a, b)$ denotes the result of raising a to the power b , where a is an integer or a finite field element and b is an integer. This may also be denoted by a^b .

NOTE—Throughout this main document, integers and field elements are denoted with lowercase letters whereas octet strings and elliptic curve points are denoted with uppercase letters.

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—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 elements of these finite fields are to be represented for use with the primitives in 5.6. As noted in 5.2, the internal representation of objects is left to the implementation and may be different. If the internal representation is different, then 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 a prime, then there is a unique (up to isomorphism) field $GF(p)$ with p elements, and 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 Annex A.

5.3.2 Odd characteristic extension fields

An odd characteristic extension field is a finite field whose number of elements is a power of an odd prime. For a positive integer k , there is a unique (up to isomorphism) field $GF(p^k)$ with p^k elements. For the purposes of conversion, the elements of $GF(p^k)$ shall be represented in a polynomial basis. For the purposes of this standard, the representation is determined by choosing a suitable irreducible binomial $f(x)$ of degree k/d over $GF(p^d)$ for some exact divisor d of k . Then, $GF(p^k)$ is isomorphic to $GF(p^d)[x]/(f(x))$ and the elements of $GF(p^k)$ can be represented as a vector

$$(a_{k-1}, a_{k-2}, \dots, a_2, a_1, a_0)$$

of $k = m/d$ elements of the field $GF(p^d)$ that we identify with the polynomial

$$a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_2x^2 + a_1x + a_0$$

where $0 \leq a_i \leq p-1$ for $0 \leq i \leq k-1$. The towering method used is represented in the form $a/b/c/\dots$, where a, b, c , etc. are integers whose product is k . For example, for $k=12$, a $3/2/2$ representation means that an element in $GF(p^k)$ is represented as a cubic extension, over a quadratic extension, over another quadratic extension over the base field. The values a_i are then read from left to right as the values at the bottom of this tower, from most significant to least significant.

The constants required by the irreducible polynomials that define the tower of extensions also form a part of the extension field representation.

A description of the arithmetic of $GF(p^k)$ is given in Annex A.

5.3.3 Binary finite fields

There is more than one way to implement multiplication in $GF(2^k)$. To specify a multiplication rule, one chooses a basis representation for the field. The basis representation is a rule for interpreting each bit string; the multiplication rule follows from this interpretation. There are two common families of basis representations: polynomial basis representations and normal basis representations. More details about these representations can be found in Annex A.

In a polynomial basis representation, each element of $GF(2^k)$ is represented by a different binary polynomial of degree less than k . More explicitly, the bit string $(a_{k-1}, \dots, a_2, a_1, a_0)$ is taken to represent the binary polynomial

$$a_{k-1}x^{k-1} + \dots + a_2x^2 + a_1x + a_0$$

The polynomial basis is the set

$$B = \{x^{k-1}, \dots, x^2, x, 1\}$$

The addition of bit strings corresponds to addition of binary polynomials.

Multiplication is defined in terms of an irreducible binary polynomial of degree m , called the field polynomial for the representation. The product of two elements is the product of the corresponding polynomials, reduced modulo $p(x)$.

A normal basis for $GF(2^k)$ is a set of the form

$$B = \{\theta, \theta^2, \theta^{2^2}, \dots, \theta^{2^{k-1}}\}$$

with the property that no subset of B adds to 0. (In the language of linear algebra, the elements of B are said to be *linearly independent*.) There exist normal bases for $GF(2^k)$ for every positive integer m .

The representation of $GF(2^k)$ via the normal basis B is carried out by interpreting the bit string $(a_0, a_1, a_2, \dots, a_{k-1})$ as the element

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

All of the elements θ^i of a normal basis B satisfy the same irreducible binary polynomial $p(x)$. This polynomial is called the field polynomial for the basis. An irreducible binary polynomial is called a normal polynomial if it is the field polynomial for a normal basis.

A description of the arithmetic of $GF(2^k)$ is given in Annex A.

5.3.4 Ternary finite fields

Ternary finite fields $GF(3^k)$ and operations on elements of $GF(3^k)$ is defined in 5.3.2.

5.3.5 Unitary extension fields

An extension field element that is unitary can be compressed in a way that enables accurate exponentiation of the important parts of the element. In a compressed form, however, such elements can still be accurately exponentiated using special algorithms. The result of pairing calculations described in this standard is always unitary, and hence, compression of the pairing value is always possible and may be desirable. If the embedding degree is a multiple of 2, then compression by a factor 2 is possible, and the Lucas function (LUC method) of exponentiation can be used (Menezes et al. [B115]). If the embedding degree is a multiple of 3, then compression by a factor of 3 is possible, and the efficient and compact subgroup trace representation (XTR) method of exponentiation can be used (Menezes et al. [B115]).

The compressed representation will either be the $GF(p^{k/2})$ or the $GF(p^{k/3})$ trace of an element in $GF(p^k)$.

See Annex A for definitions of the term unitary and for a description of the trace function.

5.4 Elliptic curves and points

An elliptic curve group $E(GF(q))$ is a set of points of the form $P = (x_p, y_p)$ where x_p and y_p are elements of $GF(q)$ that satisfy a certain equation, together with the point at infinity denoted by $\mathcal{O}$. See Annex A for more on elliptic curves and elliptic curve arithmetic.

5.5 Pairings

All pairings in this standard use groups of points in an elliptic curve. Additive notation will always be used for operations on elliptic curve points and multiplicative notation will always be used for operations on points in the multiplicative group of a finite field. Thus, the notation used for a pairing will be as follows:

$$e(aP, bQ) = e(P, Q)^{ab}$$

This notation will be used for a modified and reduced pairing as described in Annex A.

5.6 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 in this subclause; rather, it is discussed in the informative Annex E. Figure 1 shows the primitives presented in this subclause and their relationships.

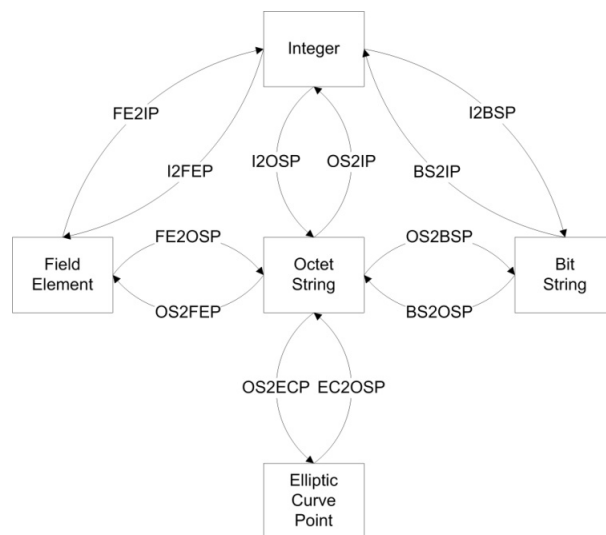


Figure 1—Data type conversion primitives

5.6.1 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 x shall be written in its unique l -digit representation base 2:

$$x = x_{l-1}2^{l-1} + x_{l-2}2^{l-2} + \dots + x_12 + x_0$$

where each 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 $b_1b_2 \dots b_l$.

For example, the integer 10 945 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.

5.6.2 Converting between bit strings and octet strings: BS2OSP and OS2BSP

To represent a bit string as an octet string, one simply pads enough zeroes on the left to make the number of bits a multiple of 8 and then breaks it up into octets. More precisely, a bit string $b_{l-1}b_{l-2} \dots b_1b_0$ of length l shall be converted to an octet string $M_{d-1}M_{d-2} \dots M_1M_0$ of length $d = \lceil l/8 \rceil$ as follows. For $0 \leq i < (d-1)$, let the octet $M_i = b_{8i+7}b_{8i+6} \dots b_{8i+1}b_{8i}$. The leftmost octet M_{d-1} shall have its leftmost $8d-l$ bits set to 0 and 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. It shall output the bit string if $d = \lceil l/8 \rceil$ and if the leftmost $8d-l$ bits of the leftmost octet are zero; it shall output “error” otherwise.

5.6.3 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 as follows:

$$x = x_{l-1}256^{l-1} + x_{l-2}256^{l-2} + \dots + x_1256 + x_0$$

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 $0 \leq i < (l-1)$. The octet string shall be $M_{l-1}M_{l-2} \cdots M_1M_0$.

For example, the integer 10 945 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.

5.6.4 Converting between finite field elements and octet strings: FE2OSP and OS2FEP

An element x of a finite field $GF(q)$, for purposes of conversion, is represented by an integer if q is an odd prime or a power of an odd prime (see 5.3.1, 5.3.2, and 5.3.5). If q is an odd prime or an odd prime power, then to represent x as an octet string, I2OSP shall be used with the integer value representing x and the length $\lceil \log_{256} q \rceil$ as inputs.

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 , the field size q , and both the field characteristic p and the extension degree k if q is an odd prime-power as inputs, and outputs the corresponding octet string.

To convert an octet string back to a field element, if q is an odd prime or an odd prime power, then OS2IP shall be used with the octet string as the input. 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.”

5.6.5 Converting between finite field elements and integers: FE2IP and I2FEP

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

- a) Convert the element α to an octet string using FE2OSP.
- b) Convert the resulting octet string to an integer i using OS2IP.
- c) Output i .

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

To convert an integer to an element of the finite field $GF(q)$, the primitive integer to finite field element primitive, or I2FEP, is used. A positive integer i with $0 \leq i < q$ shall be converted to an element α of a finite field $GF(q)$ by the following, or an equivalent, procedure:

- a) Convert the integer i to an octet string using I2OSP.
- b) Convert the resulting octet string to a field element α using OS2FEP.
- c) Output α .

5.6.6 Converting between elliptic curve points and octet strings

An elliptic curve point P (which is not the point at infinity $\mathcal{O}$) can be represented in either compressed or uncompressed form. (For internal calculations, it may be advantageous to use other representations; e.g., the projective coordinates of A.6.4.3. Also see A.6.4.2 for more information on point compression.) The uncompressed form of P is simply given by its two coordinates. The compressed form is presented in 5.6.6.1. The octet string format is defined to support both compressed and uncompressed points.

5.6.6.1 Compressed elliptic curve points

The compressed form of an elliptic curve point $P \neq \mathcal{O}$ defined over $GF(p)$ is the pair (x_p, y_p) , where x_p is the x coordinate of P and y_p is a bit that is computed as defined in 5.6.6.1.1.

Two compressed forms are defined: A least significant bit (LSB) compressed form is defined for elliptic curves over $GF(p)$, and a SORT (sorted order) compressed form is defined for elliptic curves over $GF(p^m)$. Note that the case where a curve is defined over $GF(3^m)$ can be handled the same way that a curve defined over $GF(p^m)$ can be.

5.6.6.1.1 LSB compressed form

For $E/ GF(p)$, the LSB compressed form of $(x_p, \hat{y}_p)$ has $y_p = \text{FE2IP}(\hat{y}_p) \bmod 2$. In other words, y_p is the rightmost bit of $\hat{y}_p$.

Procedures for point decompression (i.e., recovering $\hat{y}_p$ given x_p and y_p) are given in A.7.

NOTE—The term “LSB” refers to the fact that the compressed bit y_p is the least significant bit of the integer representation of $\hat{y}_p$.

5.6.6.1.2 SORT compressed form

Let (x_p, y'_p) be the inverse of the point (x_p, y_p) , where $y'_p = -y_p$.

The SORT compressed form has $\tilde{y}_p = 1$ if $\text{FE2IP}(y_p) > \text{FE2IP}(y'_p)$ and $\tilde{y}_p = 0$ otherwise.

A procedure for point decompression is given in A.7.

NOTE 1—It may be more efficient to determine $\tilde{y}_P$ by comparing the coefficients of y_P and y'_P directly, rather than first computing $\text{FE2IP}(y_P)$ and $\text{FE2IP}(y'_P)$.

NOTE 2—Although the SORT compressed form is defined here for any field, in the representations in 5.6.6.2, it is only employed for elliptic curves over $GF(p^m)$.

NOTE 3—The name “SORT” refers to the fact that the compressed bit is based on comparing the integer representations of y_P and y'_P .

5.6.6.2 Two-coordinate point representations

For all the conversion primitives in this subclause, the point $\mathcal{O}$ shall be represented by an octet string containing a single 0 octet. The rest of this subclause discusses octet string representation of a point $P \neq \mathcal{O}$. Let the x coordinate of P be x_P and the y coordinate of P be y_P . Let $(x_P, \tilde{y}_P)$ be the compressed representation of P in one of the forms in 5.6.6.1.

The representations in this subclause are all “lossless”; i.e., the elliptic curve point can be uniquely recovered from its octet string representation because both coordinates are represented.

An octet string PO representing P shall have one of the following three formats: compressed, uncompressed, or hybrid. (The hybrid format contains information of both compressed and uncompressed form.) For all primitives in this subclause, PO shall have the following general form:

$$PO = PC \parallel X \parallel Y$$

where PC is a single octet of the form $0000SUC\tilde{Y}$ defined as follows:

- Bit S is 1 if the format uses the SORT compressed form; 0 otherwise.
- Bit U is 1 if the format is uncompressed or hybrid; 0 otherwise.
- Bit C is 1 if the format is compressed or hybrid; 0 otherwise.
- Bit $\tilde{Y}$ is equal to the bit if the format is compressed or hybrid; 0 otherwise.
- X is the octet string of length $\lceil \log_{256} q \rceil$ representing x_P according to FE2OSP (see 5.6.4).
- Y is the octet string of length $\lceil \log_{256} q \rceil$ representing y_P of P according to FE2OSP (see 5.6.4) if the format is uncompressed or hybrid; Y is an empty string if the format is compressed.

The primitive that converts elliptic curve points to octet strings for a given representation is called the elliptic curve point to octet string conversion primitive—R, or EC2OSP—R, where R is the representation. It takes an elliptic curve point P and the size q of the underlying field as input and outputs the corresponding octet string PO .

The primitive that converts octet strings to elliptic curve points is called the octet string to elliptic curve point conversion primitive—r, or OS2ECP—R. It takes the octet string and the field size q as inputs and outputs the corresponding elliptic curve point, or “error.” It shall use OS2FEP to get x_P . It shall use OS2FEP to get y_P if the format is uncompressed and may output “error” if the recovered point is not on the elliptic curve. It shall use point decompression (see A.7) to get y_P if the format is compressed. It can get y_P by either of these two means if the format is hybrid, and if the format is hybrid, then it may output “error” if different values are obtained by the two means. It shall output “error” in the following cases:

- If the first octet is not as expected for the representation
- If the octet string length is not as expected for the representation
- If an invocation of OS2FEP outputs “error”
- If an invocation of the point decompression algorithm outputs “error”

The pairs of primitives for each of five representations are defined in 5.6.6.2.1 through 5.6.6.2.5.

5.6.6.2.1 Uncompressed representation: EC2OSPXY and OS2ECPXY

This representation is defined for elliptic curves over all finite fields in this standard.

In this representation, the octet PC shall have binary value 0000 0100 and the octet strings X and Y shall represent x_p and y_p , respectively. The length of the octet string PO shall be $1 + \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXY and OS2ECPXY.

5.6.6.2.2 LSB compressed representation: EC2OSPXL and OS2ECPXL

This representation is defined for elliptic curves over $GF(p)$ only.

In this representation, the octet PC shall have binary value 0000 001 $Y_{\sim}$, where $Y_{\sim}$ is equal to the bit y_p in the LSB compressed form, the octet string X shall represent x_p , and the octet string Y shall be the empty string. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXL and OS2ECPXL.

5.6.6.2.3 SORT compressed representation: EC2OSPXS and OS2ECPXS

This representation is defined for elliptic curves over $GF(p^m)$ only.

In this representation, the octet PC shall have binary value 0000 101 $Y_{\sim}$, where $Y_{\sim}$ is equal to the bit y_p in the SORT compressed form, the octet string X shall represent x_p , and the octet string Y shall be the empty string. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXS and OS2ECPXS.

5.6.6.2.4 LSB hybrid representation: EC2OSPXYL and OS2ECPXYL

This representation is defined for elliptic curves over $GF(p)$ only.

In this representation, the octet PC shall have binary value 0000 011 $Y_{\sim}$, where $Y_{\sim}$ is equal to the bit in the LSB compressed form, and the octet strings X and Y shall represent x_p and y_p , respectively. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXYL and OS2ECPXYLSORT hybrid representations: EC2OSPXYLS and OS2ECPXYLS.

5.6.6.2.5 SORT hybrid representation: EC2OSPXYs and OS2ECPXYs

This representation is defined for elliptic curves over $GF(p^m)$ only.

In this representation, the octet PC shall have binary value 0000 111 $Y_{\sim}$, where $Y_{\sim}$ is equal to the bit $y_{\sim}P$ in the SORT compressed form, and the octet strings X and Y shall represent x_p and y_p , respectively. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXYs and OS2ECPXYs.

5.6.6.3 X-coordinate-only representation: EC2OSPX and OS2ECPX

The x -coordinate-only representation in this subclause is “lossy”; i.e., the elliptic curve point *cannot* be uniquely recovered from its octet string representation because only the x coordinate is represented.

This representation is defined for elliptic curves over all fields in this standard.

In this representation, the octet PC shall have binary value 0000 0001, the octet string X shall represent x_p , and the octet string Y shall be empty. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPX and OS2ECPX.

OS2ECPX may output any of the (at most two) elliptic curve points with the given x coordinate. Thus, the original y coordinate is not necessarily recovered.

This representation should be employed only if the recipient of the octet string PO does not need to resolve the ambiguity in the y coordinate or can do so by other means.

NOTE—In some situations, only the x coordinate is needed. For instance, a shared secret value computed may only be on the x coordinate of other party’s public key, not on the y coordinate. If this representation is employed in such a situation, then when the “octet-string-to-point” conversion primitive is called, the implementation need not compute a y coordinate at all (although it may output “error” if no point exists with the given x coordinate).

5.6.6.4 Summary of representations

Table 2 summarizes the point representations in 5.6.6.2 and 5.6.6.3.

Table 2—Elliptic curve point representations

Representation	Primitives	PC	X	Y	Finite fields
Uncompressed	EC2OSPXY OS2ECPXY	0000 0100	x_p	y_p	All
LSB compressed	EC2OSPXL OS2ECPXL	0000 001 Y~	x_p	Empty	$GF(p)$
SORT compressed	EC2OSPXS OS2ECPXS	0000 101 Y~	x_p	Empty	$GF(p^m)$
LSB hybrid	EC2OSPXYL OS2ECPXYL	0000 011 Y~	x_p	y_p	$GF(p)$
SORT hybrid	EC2OSPXYs OS2ECPXYs	0000 111 Y~	x_p	y_p	$GF(p^m)$
x-coordinate-only	EC2OSPX OS2ECPX	0000 0001	x_p	Empty	All
Point $\mathcal{O}$	All	0000 0000	Empty	Empty	All

NOTE 1—The first four bits of the first octet *PC* are reserved and may be used in future formats defined in an amendment to, or in future versions of, this standard. It is essential that these bits be set to 0 and checked for 0 to distinguish the formats defined here from other formats. Of course, implementations may support other nonstandard formats that employ the reserved bits, but these formats would not conform with the ones defined in this subclause.

NOTE 2—The various representations employ distinct values for the first octet *PC*, so the octet strings produced by the different representations are nonoverlapping, except at the point $\mathcal{O}$. Consequently, it is possible to construct a generic OS2ECP primitive that handles all representations.

6. Hashing primitives

The schemes described in 7.2 through 7.4 require the use of two or more cryptographic hash functions. The schemes described in these subclauses are based on one of the hash functions defined by NIST in Federal Information Processing Standards Publication 180-4 [B60], which will be denoted *H*. The hash function *H* chosen will depend on a security parameter that defines the level of bit strength that is required. This bit strength is required to be one of the standard levels: 80, 112, 128, 192, or 256. The hash function *H* is then used to construct the required hash function as described in 6.1 through 6.4.

6.1 Hashing to an integer

The function IHF1 is a cryptographic hash function that hashes a string to an integer. The function IHF1 uses the SHA family (Federal Information Processing Standards Publication 180-4 [B60]) to accomplish this. Other hash functions can be constructed as needed.

6.1.1 The function of IHF1

Returns an integer between 0 and $n-1$ that is based on a cryptographic hash function applied to an input string.

Input:

- A string $s \in \{0,1\}^*$
- An integer n

— A security parameter $t \in \{80, 112, 128, 192, 256\}$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function. The integer n has the property that $n \leq 2^{2t}$.

Output:

— An integer $v \in Z_n$.

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) If $t = 80$, then let $H = \text{SHA-1}$.
- b) Else if $t = 112$, then let $H = \text{SHA-224}$.
- c) Else if $t = 128$, then let $H = \text{SHA-256}$.
- d) Else if $t = 192$, then let $H = \text{SHA-384}$.
- e) Else if $t = 256$, then let $H = \text{SHA-512}$.
- f) Let $v_0 = 0$.
- g) Let $h_0 = 0x00 \dots 00$, string of null bytes of length $2t$ bits.
- h) Let $t_1 = h_0 \parallel s$.
- i) Let $h_1 = H(t_1)$.
- j) Let $a_1 = \text{OS2IP}(h_1)$.
- k) Let $v_1 = a_1$.
- l) Let $t_2 = h_1 \parallel s$.
- m) Let $h_2 = H(t_2)$.
- n) Let $a_2 = \text{OS2IP}(h_2)$.
- o) Let $v_2 = 2^{2t} v_1 + a_2$.
- p) Output $v_2 \bmod n$.

6.2 Hashing to a string

The function IHF1 is a cryptographic hash function that hashes a string to a string. The function SHF1-SHA uses the SHA-1 and SHA-2 family (Federal Information Processing Standards Publication 180-4 [B60]) to accomplish this. Other hash functions can be constructed as needed.

6.2.1 Function of SHF1

Returns an n -bit string that is based on a cryptographic hash function applied to an input string.

Input:

- A string $s \in \{0,1\}^*$
- An integer n
- A security parameter $t \in \{80,112,128,192,256\}$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function. The integer n has the property that $n \leq 2^{2t}$.

Output:

- A string $v \in \{0,1\}^n$

Operation: Use the following steps:

- a) Output $\text{IHFI}(s, 2^n, t)$.

6.3 Hashing to a point in a subgroup

6.3.1 General

The function PHF-SS is a cryptographic hash function that hashes a string into a subgroup of points $E(GF(q))[p]$ on two particular supersingular elliptic curves. PHF-GFP is a cryptographic hash function that hashes a string into a subgroup of points $E(GF(q^e))[p]$, where $q > 3$. PHF-GF2 is a cryptographic hash function that hashes a string into a subgroup of points $E(GF(2^e))[p]$. PHF-GF3 is a cryptographic hash function that hashes a string into a subgroup of points $E(GF(3^e))[p]$.

6.3.2 Function of PHF-SS

Returns an element of an elliptic curve group $E(GF(q))[p]$ for a supersingular elliptic curve $E/GF(q): y^2 = x^3 + 1$ or $E/GF(q): y^2 = x^3 + x$.

Input:

- A string $s \in \{0,1\}^*$
- A security parameter $t \in \{80,112,128,192,256\}$
- A flag j taking the values 0 or 1 that defines a supersingular elliptic curve, with $j = 0$ representing the elliptic curve $E/GF(q): y^2 = x^3 + 1$ and $j = 1$ representing the elliptic curve $E/GF(q): y^2 = x^3 + x$
- A prime q with $q \equiv 11 \pmod{12}$ that defines the finite field $GF(q)$
- A prime p with $p \mid \#E(GF(q))$ and $p^2 \nmid \#E(GF(q))$ for the elliptic curve E defined by the flag j

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function. The prime q satisfies $q \equiv 11(\bmod 12)$.

Output:

- An element of $E(GF(q))[p]$ for the selected elliptic curve.

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Let $r = (q+1)/p$.
- b) If $j = 0$, then perform the following steps:
 - 1) Let $y = \text{IHF1}(s, q, t)$.
 - 2) Let $x = (y^2 - 1)^{(2q-1)/3} \bmod q$.
 - 3) Let $Q = (x, y)$.
- c) Else if $j = 1$, perform the following steps:
 - 1) Let $x = \text{IHF1}(s, q, t)$.
 - 2) If the Jacobi symbol $(x/q) = +1$, then perform the following steps
 - i) Let $y = x^{(q+1)/4} \bmod q$.
 - ii) Let $Q = (x, y)$.
 - 3) Else perform the following steps:
 - i) Let $y = (-x)^{(q+1)/4} \bmod q$.
 - ii) Let $Q = (-x, y)$.
- d) Return rQ .

6.3.3 Function of PHF-GFP

Returns an element of an elliptic curve group $E(GF(q^e))[p]$, where q is a prime and $q > 3$.

Input:

- A string $s \in \{0,1\}^*$
- A security parameter $t \in \{80, 112, 128, 192, 256\}$
- A prime q that defines the finite field $GF(q^e)$
- An integer e that defines the finite field $GF(q^e)$
- A prime p that defines the subgroup $E(GF(q^e))[p]$ of $E(GF(q^e))$
- An elliptic curve $E / GF(q^e): y^2 = x^3 + ax + b$
- A cofactor $r = \#E(GF(q^e)) / p$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function.

Output:

— An element of $E(GF(q^e))[p]$

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Let $i = 0$.
- b) Let $x = \text{H2FEQ}(\text{I2BSP}(i) || s, t, q, e)$.
- c) Let $t = x^3 + ax + b \in GF(q^e)$.
- d) If $t = 0$, then output $(x, 0)$ and stop.
- e) Apply the appropriate technique from A.2.5 to find α , the square root of t .
- f) If the result of step e) indicates that no square root exists, then increment i by 1 and then go to step b).
- g) Let $y_1 = \text{FE2IP}(\alpha)$.
- h) Let $y_2 = \text{FE2IP}(-\alpha)$.
- i) If $y_1 > y_2$ then let $y = -\alpha$ else let $y = \alpha$.
- j) Let $Q = (x, y)$.
- k) Return rQ .

6.3.4 Function of PHF-GF2

Returns an element of an elliptic curve group $E(GF(2^e))[p]$.

Input:

- A string $s \in \{0,1\}^*$
- A security parameter $t \in \{80, 112, 128, 192, 256\}$
- An integer e that defines the finite field $GF(2^e)$
- A prime p that defines the subgroup $E(GF(2^e))[p]$ of $E(GF(2^e))$
- An elliptic curve $E / GF(2^e) : y^2 + xy + y = x^3 + x^2 + b$
- A cofactor $r = \#E(GF(2^e)) / p$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function.

Output:

— An element of $E(GF(2^e))[p]$

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Let $i = 0$.
- b) Let $x = \text{H2FEQ}(i \parallel s, t, 2, e) \in GF(2^e)$.
- c) If $x = 0$, then output $(0, b^2)$ and stop.
- d) Let $t = x^3 + ax^2 + b$.
- e) If $t = 0$, then output $(x, 0)$ and stop.
- f) Let $u = x^{-2}t$.
- g) Apply the appropriate technique from A.4.7 to find an element z for which $z^2 + z = \beta$ or determine that none exist.
- h) If the result of step g) indicates that no such element exists, increment i by 1 and go to step b).
- i) Generate a random bit β that will be used to pick which y coordinate will be chosen for the calculated x coordinate.
- j) Let $y = (z + \beta)x$.
- k) Let $Q = (x, y)$.
- l) Return rQ .

6.3.5 Function of PHF-GF3

Returns an element of an elliptic curve group $E(GF(3^e))[p]$.

Input:

- A string $s \in \{0,1\}^*$
- A security parameter $t \in \{80, 112, 128, 192, 256\}$
- An integer e that defines the finite field $GF(3^e)$
- An elliptic curve $E/GF(3^e): y^2 = x^3 - x + b$ with $b = 1$ or $b = -1$
- A cofactor $r = \#E(GF(3^e)) / p$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function.

Output:

- An element of $E(GF(3^e))[p]$

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Let $i = 0$.
- b) Let $y = \text{H2FEQ}(i \| s, t, 3, e) \in GF(3^e)$.
- c) Let $u = y^2 - b$.
- d) If $\text{Tr}(u) \neq 0$, then increment i by 1 and go to step b).
- e) Use the method from A.4.10 to find an element z for which $z^3 - z = u$.
- f) Generate a random element $\beta \in GF(3^e)$ that will be used to select which possible x coordinate is chosen for a particular y coordinate.
- g) Let $x = z + \beta$.
- h) Let $Q = (x, y)$.
- i) Return rQ .

6.4 Hashing to an element of a finite field

6.4.1 Hashing to an element of a finite field: Function of BS2FQE

The function BS2FQE is a cryptographic hash function that hashes a string into the finite field $GF(q^k)$.

Input:

- A string $s \in \{0, 1\}^*$
- A prime q that defines the finite field $GF(q^k)$
- An integer e that defines the finite field $GF(q^k)$
- A security parameter $t \in \{80, 112, 128, 192, 256\}$

Input constraints: The string s is within the allowed range of values for inputs to the relevant hash function.

Output:

- An element of $GF(q^k)$

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) For $i = 0$ to $e-1$, do the following:
 - 1) Let $t_i = \text{IHF1}(\text{I2BS}(i) \| s, q, t)$.
 - 2) Next i .
- b) Return the field element represented by $x = t_0 + t_1x + \dots + t_{e-1}x^{e-1}$.

7. Pairing-based primitives

7.1 General

This clause describes the mathematical operations that are used to build the pairing-based schemes described in Clause 8 through Clause 11.

The primitives discussed in this clause make use of the following parameters:

- G_1 , a group of prime order p .
- G_2 , a group of prime order p .
- G_3 , a group of prime order p .
- A pairing $e_i : \begin{cases} G_1 \times G_2 \rightarrow G_3, i=1 \\ G_2 \times G_1 \rightarrow G_3, i=2 \end{cases}$. In all pairing-based primitives and components, the auxiliary parameter j is always defined to be $j = 3 - i$. Thus, if $i = 1$, then $j = 2$, and if $i = 2$, then $j = 1$.

The selection of the groups and specific pairing is left as a parameter to the schemes using these primitives. Annex A defines various recommended choices for these elements, depending on the performance and security requirements of the specific application.

7.2 SK primitives

These primitives define operations with cryptographic strength defined by reduction to the q -bilinear Diffie-Hellman inverse (q -BDHI) problem. For more information on this problem, to review the proof that reduces the problem of breaking these primitives to q -BDHI, and for information on the associated primitives, see Chen and Cheng [B41].

For all these operations, the systems parameters are assumed to be defined as follows.

The server secret is defined to be

- s , a random element of Z_p .

The public parameters are defined to be $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R)$, where

- G_1, G_2, G_3, e_i , the system parameters as defined in 7.1
- Q_i : a generator of G_i
- Q_j : a generator of G_j , defined such that either $Q_j = \phi(Q_i)$ (for a supersingular curve) or $Q_j = \phi_d(Q_i)$ (for an ordinary curve)
- R : sQ_i

7.2.1 SK: Generation (P-SK-G)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R)$
- The server secret s
- An encoded identity $M \in Z_p$

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; s is the private key corresponding to R , so that $R = sQ_i$; and M is an element of Z_p .

Output:

- The derived private key K_M , an element of G_j , or “error”

Operation: The following steps or their mathematical equivalent shall be used to compute K_M :

- a) If $M + s \equiv 0 \pmod{p}$, then output “error” and stop.
- b) Compute $t = (M + s)^{-1} \pmod{p}$.
- c) Compute $K_M = tQ_j$.
- d) Output K_M .

7.2.2 SK: Verification (P-SK-V)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R)$
- A public-key element M in Z_p
- The corresponding private key K_M

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups, and K_M is the private key associated with the public key M .

Output:

- The value “valid” if K_M is consistent with R and M and “invalid” otherwise

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Compute $T_1 = e_i(MQ_i + R, K_M)$.
- b) Compute $T_2 = e_i(Q_i, Q_j)$.
- c) If $T_1 = T_2$, then output the value “valid”; otherwise, output the value “invalid.”

NOTE—The value $T_2 = e_i(Q_i, Q_j)$ is a constant and can be computed once and cached, saving a pairing operation.

7.2.3 SK: Encryption (P-SK-E)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R)$
- The public-key element M
- A message randomizer r , an integer with $0 \leq r \leq p - 1$

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; R is the public parameter associated with the server secret s ; and M is an element of Z_p .

Output:

- A ciphertext E , where E is an element of G_i , along with a blinding factor B , an element of G_i , along with a blinding factor B , an element of G_3

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Compute $E = r(MQ_i + R)$.
- b) Compute $B = e_i(Q_i, Q_j)^r$.
- c) Output E and B .

NOTE—The blinding factor B is the same as that obtained in P-SK-D (7.2.4).

7.2.4 SK: Decryption (P-SK-D)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R)$
- The public-key element M
- The associated private key K_M
- A ciphertext E

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; R is a public key, an element of G_i ; M is an element of Z_p ; K_{ID} , an element of G_j , is a valid generated value for M according to P-SK-GV; and E is an element of G_i .

Output:

- A blinding factor B , an element of G_3 .

Operation: Use the following steps:

- a) Compute $B = e_i(E, K_M)$.
- b) Output B .

NOTE—If both parties follow the algorithms specified, then the value B computed here is $e_i(Q_1, Q_2)^r$, the same as that obtained in P-SK-E (7.2.3).

7.3 BB₁ primitives

7.3.1 General

These primitives define operations with cryptographic strength defined by reduction to the BDH problem. For more information on this problem, to review the proof that reduces the problem of breaking these primitives to BDH, and for information on the associated primitives, see Boneh and Boyen [B26].

For all these operations, the systems parameters are assumed to be defined as follows.

The server secret is defined to be

- s , comprising three random elements s_1, s_2, s_3 , of Z_p

The public parameters are defined to be $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R, T, V)$, where

- G_1, G_2, G_3, e_i : the system parameters as defined in Clause 6
- Q_i : a generator of G_i
- Q_j : a generator of G_j
- $R = s_1 Q_i$
- $T = s_3 Q_i$
- $V = e_i(R, s_2 Q_j)$

7.3.2 BB₁: Generation (P-BB1-G)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R, T, V)$
- The server secret s
- A randomizer r , an integer in Z_p
- An encoded identity M in Z_p , typically derived from an identity string

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; s is the server secret corresponding to $\mathcal{P}$; and M is an element of Z_p .

Output:

- The derived secret key, $K_{0,M}$ and $K_{1,M}$, two elements of G_j

Operation: The following steps or their mathematical equivalent shall be used to compute $K_{0,M}$ and $K_{1,M}$:

- a) Compute $t = s_1 s_2 + r(s_1 M + s_3)$ in Z_p .
- b) Compute $K_{0,M} = tQ_j$.
- c) Compute $K_{1,M} = rQ_j$.
- d) Output $K_{0,M}$ and $K_{1,M}$.

7.3.3 BB₁: Verification (P-BB1-V)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R, T, V)$
- A public-key element M in Z_p
- The corresponding key $K_{0,M}$ and $K_{1,M}$

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups and correspond to the server secret s ; M is an element of Z_p ; and $K_{0,M}$ and $K_{1,M}$ are two elements of G_j .

Output:

- The value “valid” if $K_{0,M}$ and $K_{1,M}$ are consistent with P and M and “invalid” otherwise

Operation: The following steps or their mathematical equivalent shall be used to produce correct output:

- a) Compute $T_0 = e_i(Q_i, K_{0,M})$.
- b) Compute $T_1 = e_i(MR + T, K_{1,M})$.
- c) If $T_0 = T_1V$, then output the value “valid”; otherwise, output the value “invalid.”

7.3.4 BB₁: Encryption (P-BB1-E)

Input:

- The parameters $\mathcal{P} = (G_1, G_2, G_3, e_i, Q_1, Q_2, R, T, V)$
- The public-key element M in Z_p
- A message randomizer r , an element of Z_p

Input constraints: The parameters $\mathcal{P}$ describe a valid bilinear group and the public parameters associated with the server secret s ; and M is an element of Z_p .