

1 **IEEE P1636.3™/D1**
2 **Draft Standard for Identity-based**
3 **Public-key Cryptography Using**
4 **Pairings**

5 Prepared by the P1363 Working Group of the
6 Microprocessor Standards Committee

7 Copyright © <year> by the Institute of Electrical and Electronics Engineers, Inc.
8 Three Park Avenue
9 New York, New York 10016-5997, USA

10 All rights reserved.

11 This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to
12 change. USE AT YOUR OWN RISK! Because this is an unapproved draft, this document must not be
13 utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards
14 Committee participants to reproduce this document for purposes of international standardization
15 consideration. Prior to adoption of this document, in whole or in part, by another standards development
16 organization, permission must first be obtained from the IEEE Standards Activities Department
17 (stds.ipr@ieee.org). Other entities seeking permission to reproduce this document, in whole or in part, must
18 also obtain permission from the IEEE Standards Activities Department.

19 IEEE Standards Activities Department
20 445 Hoes Lane
21 Piscataway, NJ 08854, USA

1 **Abstract:** This standard specifies common identity-based public-key cryptographic techniques
2 that use pairings, including mathematical primitives for secret value (key) derivation, public-key
3 encryption, and digital signatures, and cryptographic schemes based on those primitives. It also
4 specifies related cryptographic parameters, public keys and private keys. The purpose of this
5 standard is to provide a reference for specifications of a variety of techniques from which
6 applications may select.
7 **Keywords:** Public-key cryptography, encryption, identity-based encryption, pairing-based
8 encryption, pairing-based cryptography.
9

10

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

Copyright © 200X by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published XX Month XXXX. 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-XXXX-XXXX-X STDXXXX
Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXX

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.

1 This page is left blank intentionally.

1 Introduction

2 This introduction is not part of IEEE P1363.3/D1, Draft Standard for Identity-based Public-key Cryptography Using
3 Pairings.

4 <Select this text and type or paste introduction text>

5 Notice to users

6 Laws and regulations

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

12 Copyrights

13 This document is copyrighted by the IEEE. It is made available for a wide variety of both public and
14 private uses. These include both use, by reference, in laws and regulations, and use in private self-
15 regulation, standardization, and the promotion of engineering practices and methods. By making this
16 document available for use and adoption by public authorities and private users, the IEEE does not waive
17 any rights in copyright to this document.

18 Updating of IEEE documents

19 Users of IEEE standards should be aware that these documents may be superseded at any time by the
20 issuance of new editions or may be amended from time to time through the issuance of amendments,
21 corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the
22 document together with any amendments, corrigenda, or errata then in effect. In order to determine whether
23 a given document is the current edition and whether it has been amended through the issuance of
24 amendments, corrigenda, or errata, visit the IEEE Standards Association web site at
25 <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

26 For more information about the IEEE Standards Association or the IEEE standards development process,
27 visit the IEEE-SA web site at <http://standards.ieee.org>.

28 Errata

29 Errata, if any, for this and all other standards can be accessed at the following URL:
30 <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL
31 for errata periodically.

32

1 Interpretations

2 Current interpretations can be accessed at the following URL: [http://standards.ieee.org/reading/ieee/interp/](http://standards.ieee.org/reading/ieee/interp/index.html)
3 [index.html](http://standards.ieee.org/reading/ieee/interp/index.html).

4 Patents

5 Attention is called to the possibility that implementation of this standard may require use of subject matter
6 covered by patent rights. By publication of this standard, no position is taken with respect to the existence
7 or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying
8 Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity
9 or scope of Patents Claims or determining whether any licensing terms or conditions provided in
10 connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable
11 or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any
12 patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further
13 information may be obtained from the IEEE Standards Association.

14 Participants

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

16 **William Whyte**, *Chair*

17 **Don B. Johnson**, *Vice Chair*

18					
19	Participant1	22	Participant4	25	Participant7
20	Participant2	23	Participant5	26	Participant8
21	Participant3	24	Participant6	27	Participant9

28

29 The following members of the **[individual/entity]** balloting committee voted on this standard. Balloters
30 may have voted for approval, disapproval, or abstention.

31

32 (to be supplied by IEEE)

1	CONTENTS	
2	1. Overview	9
3	1.1 Scope	9
4	1.2 Purpose	9
5	1.3 Organization of the Document	10
6	1.3.1 Structure of the Main Document.....	10
7	1.3.2 Structure of the Annexes.....	10
8	2. References	10
9	3. Definitions	11
10	4. Types of Cryptographic Techniques.....	15
11	4.1 General Model.....	15
12	4.2 Primitives.....	15
13	4.3 Schemes.....	16
14	4.4 Additional Methods.....	17
15	4.5 Table Summary	17
16	5. Mathematical Conventions.....	19
17	5.1 Mathematical Notation	19
18	5.2 Bit Strings and Octet Strings	22
19	5.3 Finite Fields.....	22
20	5.3.1 Prime Finite Fields.....	22
21	5.3.2 Odd characteristic extension fields	23
22	5.3.3 Unitary extension fields.....	23
23	5.4 Pairings.....	24
24	5.5 Elliptic Curves and Points	25
25	5.6 Data Type Conversion.....	25
26	5.6.1 Converting Between Integers and Bit Strings: I2BSP and BS2IP.....	26
27	5.6.2 Converting Between Bit Strings and Octet Strings: BS2OSP and OS2BSP ...	26
28	5.6.3 Converting between Integers and Octet Strings: I2OSP and OS2IP	27
29	5.6.4 Converting between finite field elements and octet strings: FE2OSP and	
30	OS2FEP.....	27
31	5.6.5 Converting Finite Field Elements to Integers: FE2IP.....	28
32	5.6.6 Converting between elliptic curve points and octet strings	28
33	5.6.6.1 Compressed elliptic curve points	28
34	5.6.6.1.1 LSB compressed form.....	28
35	5.6.6.1.2 SORT compressed form.....	28
36	5.6.6.2 Two-coordinate point representations.....	29
37	5.6.6.2.1 Uncompressed representation: EC2OSP-XY and OS2ECP-XY	30
38	5.6.6.2.2 LSB compressed representation: EC2OSP-XL and OS2ECP-XL.....	30
39	5.6.6.2.3 SORT compressed representation: EC2OSP-XS and OS2ECP-XS ..	30
40	5.6.6.2.4 LSB hybrid representation: EC2OSP-XYL and OS2ECP-XYL	30
41	5.6.6.2.5 SORT hybrid representation: EC2OSP-XYS and OS2ECP-XYS.....	31
42	5.6.6.3 x -coordinate only representation: EC2OSP-X and OS2ECP-X.....	31
43	5.6.6.4 Summary of representations	31
44	6. Hashing Primitives	32

1	6.1 Hashing to an integer.....	32
2	6.1.1 The Function IHF1-SHA	32
3	6.2 Hashing to a string.....	33
4	6.2.1 The Function SHF1-SHA	33
5	6.3 Hashing to a point on a curve.....	34
6	6.3.1 The Function PHF1-SHA	34
7	7. Pairing-based Primitives.....	35
8	7.1 Pairing-based SK primitives.....	36
9	7.1.1 Pairing-based SK: Generation (P-SK-G).....	36
10	7.1.2 Pairing-based SK: Verification (P-SK-V)	37
11	7.1.3 Pairing-based SK: Encryption (P-SK-E)	37
12	7.1.4 Pairing-based SK: Decryption (P-SK-D).....	38
13	7.2 Pairing-based BB_1 primitives	38
14	7.2.1 Pairing-based BB_1 : Generation (P- BB_1 -G).....	39
15	7.2.2 Pairing-based BB_1 : Verification (P- BB_1 -V).....	39
16	7.2.3 Pairing-based BB_1 : Encryption (P- BB_1 -E).....	40
17	7.2.4 Pairing-based BB_1 : Decryption (P- BB_1 -D).....	40
18	7.3 Pairing-based BF primitives.....	41
19	7.3.1 Pairing-based BF: Generation (P-BF-G)	41
20	7.3.2 Pairing-based BF: Verification (P-BF-V).....	42
21	7.3.3 Pairing-based BF: Encryption (P-BF-E).....	42
22	7.3.4 Pairing-based BF: Decryption (P-BF-D).....	43
23	7.4 Pairing-based HIBE primitives.....	43
24	7.4.1 Pairing-based HIBE: Generation (P-HIBE-G).....	44
25	7.4.2 Pairing-based HIBE: Verification (P-HIBE-V).....	44
26	7.4.3 Pairing-based HIBE: Encryption (P-HIBE-E).....	45
27	7.4.4 Pairing-based HIBE: Decryption (P-HIBE-D)	45
28	7.5 Pairing-based HIBS primitives.....	46
29	7.5.1 Pairing-based HIBS: Generation (P-HIBS-G).....	47
30	7.5.2 Pairing-based HIBS: Verification (P-HIBS-V)	47
31	7.5.3 Pairing-based HIBS: Signature Generation (P-HIBS-SG)	48
32	7.5.4 Pairing-based HIBS: Signature Verification (P-HIBS-SV).....	48
33	7.6 Pairing-based Type1 Proxy Re-encryption primitives.....	49
34	7.6.1 Pairing-based BB_2 : Generation (P- BB_2 -G).....	50
35	7.6.2 Pairing-based BB_2 : Verification (P- BB_2 -V).....	50
36	7.6.3 Pairing-based BB_2 : Encryption (P- BB_2 -E).....	51
37	7.6.4 Pairing-based BB_2 : Decryption (P- BB_2 -D).....	51
38	7.7 Pairing-based EV primitives.....	52
39	7.7.1 Pairing-based EV: Generation (P-EV-G).....	52
40	7.7.2 Pairing-based EV: Encryption (P-EV-E).....	52
41	7.7.3 Pairing-based EV: Decryption (P-EV-D)	53
42	7.8 Pairing-based BR_1 primitives	53
43	7.8.1 Pairing-based BR_1 : Generation (P- BR_1 -G).....	53
44	7.8.2 Pairing-based BR_1 : Re-encryption (P- BR_1 -RE)	54
45	7.8.3 Pairing-based BR_1 : Verification (P- BR_1 -V).....	55
46	7.9 Pairing-based Wang key agreement primitives	55
47	7.9.1 Pairing-based Wang key agreement: derive public key (P-WKA-D1).....	55
48	7.9.2 Pairing-based Wang key agreement: derive private key (P-WKA-D2).....	56

1	7.9.3 Pairing-based Wang key agreement: derive secret value (P-WKA-D3)	56
2	7.10 Pairing-based SCK key agreement primitives	57
3	7.10.1 Pairing-based SCK key agreement: derive secret value (P-SCK-D1)	57
4	7.11 Pairing-based Type-2 Proxy Re-encryption primitives	59
5	7.11.1 Pairing-based BR2: Generation (P-BR2-G).....	59
6	7.11.2 Pairing-based BR2: Re-encryption (P-BR2-RE)	59
7	7.11.3 Pairing-based BR2: Verification (P-BR2-V).....	60
8	8. Pairing-based Identity-based Encryption Schemes.....	60
9	8.1 Operations	61
10	8.1.1 Shared operations.....	61
11	8.1.2 IBE Operations.....	61
12	8.1.3 ID-KEM Operations.....	61
13	8.2 The pairing-based SK Scheme	62
14	8.2.1 SK-KEM	62
15	8.2.1.1 SK KEM: Setup (SK-KEM-S).....	62
16	8.2.1.2 SK KEM: Extract (SK-KEM-EX)	62
17	8.2.1.3 SK KEM: Encapsulate (SK-KEM-EN)	63
18	8.2.1.4 SK KEM: Decapsulate (SK-KEM-DE)	63
19	8.3 The BB1 Scheme.....	64
20	8.3.1 BB1-KEM	64
21	8.3.1.1 BB ₁ KEM: Setup (BB1-KEM-S).....	64
22	8.3.1.2 BB ₁ KEM: Extract (BB1-KEM-EX)	64
23	8.3.1.3 BB ₁ KEM: Encapsulate (BB1-KEM-EN).....	65
24	8.3.1.4 BB ₁ KEM: Decapsulate (BB1-KEM-DE)	65
25	8.3.2 BB ₁ IBE	65
26	8.3.3 BB ₁ IBE: Setup (BB1-IBE-S).....	66
27	8.3.4 BB ₁ IBE: Extract (BB1-IBE-EX)	66
28	8.3.4.1 BB ₁ IBE: Encrypt (BB1-IBE-EN)	66
29	8.3.4.2 BB ₁ IBE: Decrypt (BB1-IBE-DE).....	67
30	8.4 The BF Scheme	67
31	8.4.1 BF IBE	67
32	8.4.1.1 BF IBE: Setup (BF-IBE-S).....	68
33	8.4.1.2 BF IBE: Extract (BF-IBE-EX).....	68
34	8.4.1.3 BF IBE: Encrypt (BF-IBE-EN)	68
35	8.4.1.4 BF IBE: Decrypt (BF-IBE-DE)	69
36	9. Pairing-based Signature Schemes.....	69
37	9.1 DHI Signature.....	69
38	9.1.1 DHI Signature: Setup (DHI-SIG-S).....	70
39	9.1.2 DHI Signature: Extract (DHI-SIG-EX)	70
40	9.1.3 DHI Signature: Create Signature (DHI-SIG-SI).....	70
41	9.1.4 DHI Signature: Verify Signature (DHI-SIG-VE)	71
42	10. Pairing-based Signcryption Schemes	72
43	10.1 DHI Signcryption	72
44	10.1.1 DHI Signcryption: Setup (DHI-SC-S).....	72
45	10.1.2 DHI Signcryption: Extract (DHI-SC-EX)	72

1	10.1.3 DHI Signcryption: Sign and Encrypt (DHI-SC-SE).....	73
2	10.1.4 DHI Signcryption: Decrypt and Verify (DHI-SC-DV)	73
3	11. Pairing-based Key Agreement Schemes.....	74
4	11.1 Wang key agreement.....	75
5	11.1.1 Wang Key Agreement: Generate Shared Secrets (W-KA-G).....	75
6	11.2 SCC Key Agreement.....	76
7	11.2.1 SCC Key Agreement: Generate Shared Secrets (SCC-KA-G).....	76
8	12. Pairing-based Proxy Re-encryption Schemes.....	77
9	12.1 M1 Proxy Re-encryption Scheme.....	77
10	12.1.1 M1 Proxy Re-encryption: Setup (M1-PRE-S).....	78
11	12.1.2 M1 Proxy Re-encryption: Extract1 (M1-PRE-EX1)	78
12	12.1.3 M1 Proxy Re-encryption: Encryption1 (M1-PRE-EN1)	79
13	12.1.4 M1 Proxy Re-encryption: Decryption1 (M1-PRE-DE1).....	79
14	12.1.5 M1 Proxy Re-encryption: Extract2 (M1-PRE-EX2)	80
15	12.1.6 M1 Proxy Re-encryption: Encryption2 (M1-PRE-EN2)	80
16	12.1.7 M1 Proxy Re-encryption: Decryption2 (M1-PRE-DE2).....	80
17	12.1.8 M1 Proxy Re-encryption: Extract3 (M1-PRE-EX3)	81
18	12.1.9 M1 Proxy Re-encryption: Re-encryption (M1-PRE-RE)	81
19	12.1.10 M1 Proxy Re-encryption: Verification (M1-PRE-VE).....	81
20	12.2 M2 Proxy Re-encryption scheme	82
21	12.2.1 M2 Proxy Re-encryption: Setup (M2-PRE-S).....	82
22	12.2.2 M2 Proxy Re-encryption: Extract (M2-PRE-EX)	83
23	12.2.3 M2 Proxy Re-encryption: Re-encryption (M2-PRE-RE)	83
24	12.2.4 M2 Proxy Re-encryption: Verification (M2-PRE-VE).....	84
25		

26 1. Overview

27 1.1 Scope

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

32 1.2 Purpose

33 The proliferation of electronic communication and the internet brings with it the need for privacy and data
 34 protection. Public key cryptography offers fundamental technology addressing this need. Many alternative
 35 public-key techniques have been proposed, each with its own benefits. IEEE Std 1363-2000 and IEEE Std
 36 1363a-2004 have produced a comprehensive reference defining a range of common public-key techniques
 37 covering key agreement, public-key encryption and digital signatures from several families, namely the
 38 discrete logarithm, integer factorization, and elliptic curve families. IEEE P1363.3 will specify identity-
 39 based cryptographic techniques based on pairings. These offer advantages over classic public key

1 techniques previously specified. Examples are the lack of a requirement to exchange or look up public keys
2 of a recipient and the simplified use of short-lived keys.

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

6 **1.3 Organization of the Document**

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

8 **1.3.1 Structure of the Main Document**

9 — Section 1 is a general overview.

10 — Section 2 provides references to other standards and publications.

11 — Section 3 defines some terms used throughout this standard.

12 — Section 4 gives an overview of the types of cryptographic techniques that are defined in this
13 standard.

14 — Section 5 describes certain mathematical conventions used in the standard, including notation and
15 representation of mathematical objects. It also defines formats to be used in communicating the
16 mathematical objects as well as primitives for data type conversion.

17 — Section 6 describes certain pairing based cryptographic primitives that are used to build the
18 complete algorithms described in subsequent sections.

19 — Section 7 defines identity-based key management schemes, which includes identity-based
20 encryption (IBE) and identity-based key encapsulation mechanisms (KEM).

21 — Section 8 defines certain auxiliary functions supporting the techniques in Sections 7.

22 **1.3.2 Structure of the Annexes**

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

25 — Annex A (Informative) Number-Theoretic Background

26 — Annex B (Normative) Conformance

27 — Annex C (Informative) Rationale

28 — Annex D (Informative) Security Considerations

29 — Annex E (Informative) Formats

30 — Annex F (Informative) Bibliography

31 **2. References**

32 This standard shall be used in conjunction with following publications:

1 Federal Information Processing Standards Publication 180-1 (FIPS 180-1), Secure Hash Standard, U.S.
 2 Department of Commerce/National Institute of Standards and Technology, National Technical Information
 3 Service, Springfield, Virginia, April 17, 1995.

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

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

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

10 NOTE 3—A bibliography is provided in Annex F.

11 3. Definitions

12 For the purposes of this draft standard, the following terms and definitions apply. *The Authoritative*
 13 *Dictionary of IEEE Standards Terms* should be referenced for terms not defined in this clause.

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

17 **bit length**: See: length.

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

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

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

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

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

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

- 1 **digital signature scheme:** a method for providing authentication. In public-key cryptography, this method
 2 can be used to generate a digital signature on a message with a private key in such a way that anyone
 3 knowing the corresponding public key can verify that the digital signature was properly produced. See
 4 also: digital signature; public key; public-key cryptography; private key; signature scheme with appendix;
 5 signature scheme with message recovery.
- 6 **domain parameters:** a set of mathematical objects, such as fields or groups, and other information,
 7 defining the context in which public/private key pairs exist. More than one key pair may share the same
 8 domain parameters. Not all cryptographic families have domain parameters. See also: public/private key
 9 pair; valid domain parameters.
- 10 **domain parameter validation:** the process of ensuring or verifying that a set of domain parameters is
 11 valid. See also: domain parameters; key validation; valid domain parameters.
- 12 **encrypt:** to produce ciphertext from plaintext. Contrast: decrypt. See also: ciphertext; encryption;
 13 plaintext.
- 14 **encryption scheme:** a method for providing privacy. In public-key cryptography, this method can be used
 15 to modify a message with the help of a public key to produce what is known as ciphertext in such a way
 16 that only the holder of the corresponding private key can recover the original message from the ciphertext.
 17 See also: ciphertext; plaintext; private key; public key; public-key cryptography.
- 18 **family:** See: cryptographic family.
- 19 **field:** a setting in which the usual mathematical operations (addition, subtraction, multiplication, and
 20 division by nonzero quantities) are possible and obey the usual rules (such as the commutative, associative,
 21 and distributive laws). A DL or EC scheme is always based on computations in a field. See [Kob94] for a
 22 precise mathematical definition.
- 23 **finite field:** a field in which there are only a finite number of quantities. The DL and EC schemes are
 24 always implemented over finite fields. See Section 5 for a description of the particular finite fields used in
 25 this standard.
- 26 **first bit:** the leading bit of a bit string or an octet. For example, the first bit of 0110111 is 0. Contrast: last
 27 bit. Syn: most significant bit; leftmost bit. See also: bit string; octet.
- 28 **first octet:** the leading octet of an octet string. For example, the first octet of 1c 76 3b e4 is 1c. Contrast:
 29 last octet. Syn: most significant octet; leftmost octet. See also: octet; octet string.
- 30 **key agreement:** a method by which two entities, using each other's public keys and their own private keys,
 31 agree on a common secret key that only they know. The secret key is then commonly used in some
 32 symmetric cryptography technique. See also: private key; public key; secret key; symmetric cryptography.
- 33 **key confirmation:** the assurance provided to each party participating in a key agreement protocol that the
 34 other party is capable of computing the agreed-upon key, and that it is the same for both parties (see
 35 D.5.1.3 for more information). See also: key agreement.
- 36 **key derivation:** the process of deriving a secret key from a secret value. See also: secret key; secret value.
- 37 **key pair:** See: public/private key pair.
- 38 **key validation:** the process of ensuring or verifying that a key or a key pair is valid. See also: domain
 39 parameter validation; public/private key pair; valid key; valid key pair.

- 1 **last bit**: The trailing bit of a bit string or an octet. For example, the last bit of 0110111 is 1. Contrast: first
2 bit. Syn: least significant bit; rightmost bit. See also: first bit; octet.
- 3 **last octet**: The trailing octet of an octet string. For example, the last octet of 1c 76 3b e4 is e4. Contrast:
4 first octet. Syn: least significant octet; rightmost octet. See also: octet; octet string.
- 5 **least significant**: See: last bit; last octet.
- 6 **length**: (1) Length of a bit string is the number of bits in the string. (2) Length of an octet string is the
7 number of octets in the string. (3) Length in bits of a nonnegative integer n is $\lceil \log_2(n+1) \rceil$ (i.e., the
8 number of bits in the integer's binary representation). (4) Length in octets of a nonnegative integer n is
9 $\lceil \log_{256}(n+1) \rceil$ (i.e., the number of digits in the integer's representation base 256). For example, the length
10 in bits of the integer 500 is 9, whereas its length in octets is 2.
- 11 **leftmost bit**: See: first bit.
- 12 **leftmost octet**: See: first octet.
- 13 **message recovery**: See: signature with message recovery.
- 14 **message representative**: a mathematical value for use in a cryptographic primitive, computed from a
15 message that is input to an encryption or a digital signature scheme. See also: encryption scheme; digital
16 signature scheme.
- 17 **most significant**: See: first bit; first octet.
- 18 **octet**: A bit string of length 8. An octet has an integer value between 0 and 255 when interpreted as a
19 representation of an integer in base 2. An octet can also be represented by a hexadecimal string of length 2,
20 where the hexadecimal string is the representation of its integer value base 16. For example, the integer
21 value of the octet 10011101 is 157; its hexadecimal representation is 9d. See also: bit string.
- 22 **octet string**: An ordered sequence of octets. See also: octet.
- 23 **parameters**: See: domain parameters.
- 24 **plaintext**: a message before encryption has been applied to it; the opposite of ciphertext. Contrast:
25 ciphertext. See also: encryption.
- 26 **private key**: the private element of the public/private key pair. See also: public/private key pair; valid key.
- 27 **public key**: the public element of the public/private key pair. See also: public/private key pair; valid key.
- 28 **public-key cryptography**: methods that allow parties to communicate securely without having prior
29 shared secrets, usually through the use of public/private key pairs. Contrast: symmetric cryptography. See
30 also: public/private key pair.
- 31 **public/private key pair**: a pair of cryptographic keys used in public-key cryptography, consisting of a
32 public key and a private key that correspond to each other by some mathematical relation. The public key
33 is commonly available to a wide audience and can be used to encrypt messages or verify digital signatures;
34 the private key is held by one entity and not revealed to anyone--it is used to decrypt messages encrypted
35 with the public key and/or produce signatures that can be verified with the public key. A public/private key
36 pair can also be used in key agreement. In some cases, a public/private key pair can only exist in the
37 context of domain parameters. See also: digital signature; domain parameters; encryption; key agreement;
38 public-key cryptography; valid key; valid key pair.

- 1 **rightmost bit**: See: last bit.
- 2 **rightmost octet**: See: last octet.
- 3 **root**: if $f(x)$ is a polynomial, then its root is a value r of the variable x such that $f(r) = 0$.
- 4 **secret key**: a key used in symmetric cryptography; commonly needs to be known to all parties involved,
5 but cannot be known to an adversary. Contrast: public/private key pair. See also: key agreement; shared
6 secret key; symmetric cryptography.
- 7 **secret value**: a value that can be used to derive a secret key, but typically cannot by itself be used as a
8 secret key. See also: secret key.
- 9 **shared secret key**: a secret key shared by two parties, usually derived as a result of a key agreement
10 scheme. See also: key agreement; secret key.
- 11 **shared secret value**: a secret value shared by two parties, usually during a key agreement scheme. See
12 also: key agreement; secret value.
- 13 **signature**: See: digital signature.
- 14 **signature scheme with appendix**: a digital signature scheme that requires the signed message as input to
15 the verification algorithm. Contrast: signature scheme with message recovery. See also: digital signature;
16 digital signature scheme.
- 17 **signature scheme with message recovery**: a digital signature scheme that contains enough information for
18 recovery of the signed message, thus limiting the possible message size while eliminating the need to
19 transmit the message with the signature and input it to the verification algorithm. Contrast: signature
20 scheme with appendix. See also: digital signature; digital signature scheme.
- 21 **signature verification**: the process of verifying a digital signature. See also: digital signature; digital
22 signature scheme.
- 23 **symmetric cryptography**: methods that allow parties to communicate securely only when they already
24 share some prior secrets, such as the secret key. Contrast: public-key cryptography. See also: secret key.
- 25 **valid domain parameters**: a set of domain parameters that satisfies the specific mathematical definition
26 for the set of domain parameters of its family. While a set of mathematical objects may have the general
27 structure of a set of domain parameters, it may not actually satisfy the definition (for example, it may be
28 internally inconsistent) and thus not be valid. See also: domain parameters; public/private key pair; valid
29 key; valid key pair; validation.
- 30 **valid key**: a key (public or private) that satisfies the specific mathematical definition for the keys of its
31 family, possibly in the context of its set of domain parameters. While some mathematical objects may have
32 the general structure of keys, they may not actually lie in the appropriate set (for example, they may not lie
33 in the appropriate subgroup of a group or be out of the bounds allowed by the domain parameters) and thus
34 not be valid keys. See also: domain parameters; public/private key pair; valid domain parameters; valid
35 key pair; validation.
- 36 **valid key pair**: a public/private key pair that satisfies the specific mathematical definition for the key pairs
37 of its family, possibly in the context of its set of domain parameters. While a pair of mathematical objects
38 may have the general structure of a key pair, the keys may not actually lie in the appropriate sets (for
39 example, they may not lie in the appropriate subgroup of a group or be out of the bounds allowed by the
40 domain parameters) or may not correspond to each other; such a pair is thus not a valid key pair. See also:
41 domain parameters; public/private key pair; valid domain parameters; valid key; validation.

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

2 **4. Types of Cryptographic Techniques**

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

6 **4.1 General Model**

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

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

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

- 21 — Primitives – basic mathematical operations that are based on number-theoretic hard problems.
22 Primitives are not meant to achieve security just by themselves, but they serve as building blocks
23 for schemes.
- 24 — Schemes – a collection of related operations combining primitives and additional methods (Section
25 4.4). Schemes can provide complexity-theoretic security which is enhanced when they are
26 appropriately applied in protocols.
- 27 — Protocols – sequences of operations to be performed by multiple parties to achieve some security
28 goal. Protocols can achieve desired security for applications if implemented correctly.

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

33 A general framework of primitives is provided in Sections 5, 6, and 7; specific schemes are defined in
34 Section 8. This standard, however, does not define protocols. These are application-specific and hence are
35 outside the scope of the standard. Nevertheless, the techniques defined in this standard are key components
36 for constructing various cryptographic protocols. Also, Annex D discusses security considerations related
37 to how the techniques can be used in protocols to achieve certain security attributes.

38 **4.2 Primitives**

39 The following primitives are defined in this standard:

- 1 — Pairing-based Diffie-Hellman, in randomized and non-randomized formulations
- 2 — Pairing-based commutative blinding
- 3 — Pairing-based full domain hash

4 Each of these primitive types has four component primitives:

- 5 — Generation. This is typically used to extract a private key at a key server
- 6 — Verification of generated value
- 7 — Encryption
- 8 — Decryption

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

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

19 The specification of a primitive consists of the following information:

- 20 — Input to the primitive
- 21 — Assumptions about the input made in the description of the operation performed by the primitive
- 22 — Output from the primitive
- 23 — Operation performed by the primitive, expressed as a series of steps
- 24 — Conformance region recommendations describing the minimum recommended set of inputs for
25 which an implementation should operate in conformance with the primitive (see Annex B for more
26 on conformance)

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

30 **4.3 Schemes**

31 The following types of schemes are defined in this standard.

- 32 — Identity-based Encryption
- 33 — Identity-based Key Encapsulation
- 34 — Identity-based Signatures
- 35 — Identity-based Sigcryption

36 The goal of these schemes is to allow encrypted communication between multiple parties assuming the
37 presence of one or more trusted key servers. These schemes allow the sending party to transform a string
38 representing the identity of the receiving party a set of key server parameters into a public key. This public

1 key can then be used to encrypt or derive a symmetric key, typically used with a symmetric encryption
 2 algorithm to encrypt a message to the second party. The receiving party must then request a private key for
 3 that identity string from a key server. The key server uses a derivation operation to calculate the receiver's
 4 private key, which is communicated back to the receiver. Subsequent communications to the receiver
 5 which use the same identity string can be decrypted with a stored version of this key.

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

9 Schemes also include key management operations, such as selecting a private key or obtaining another
 10 party's public key. For proper security, a party needs to be assured of the true owners of the keys and
 11 domain parameters and of their validity. Generation of domain parameters and keys needs to be performed
 12 properly, and in some cases validation also needs to be performed. While outside the scope of this
 13 standard, proper key management is essential for security. It is addressed in more detail in Annex D.

14 The specification of a scheme consists of the following information:

- 15 — Scheme options, such as choices for primitives and additional methods
- 16 — One or more operations, depending on the scheme, expressed as a series of steps
- 17 — Conformance region recommendations for implementations conforming with the scheme (see
 18 Annex B for more on conformance)

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

22 **4.4 Additional Methods**

23 **4.5 Table Summary**

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

26 **Table 1 Summary of schemes and their components.**

Scheme Name	Components	Primitives
SK-KEM	SK-KEM-S	P-SK-G
	SK-KEM-EX	P-SK-V
	SK-KEM-EN	P-SK-E
	SK-KEM-DE	P-SK-D
		IHF1 SHF1
BB1-KEM	BB1-KEM-S	P-BB1-G

	BB1-KEM-EX BB1-KEM-EN BB1-KEM-DE	P-BB1-V P-BB1-E P-BB1-D IHF1 SHF1
BB1-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 PHF1
DHI-Signature	DHI-SIG-S DHI-SIG-EX DHI-SIG-SI DHI-SIG-VE	
DHI-Signcryption	DHI-SC-S DHI-SC-EX DHI-SC-SE DHI-SC-DV	
Wang key agreement		

SCC key agreement		
M1-PRE	M1-PRE-S M1-PRE-EX1 M1-PRE-EN1 M1-PRE-DE1 M1-PRE-EX2 M1-PRE-EN2 M1-PRE-DE2 M1-PRE-EX3 M1-PRE-RE M1-PRE-VE	
M2-PRE	M2-PRE-S M2-PRE-EX M2-PRE-RE M2-PRE-VE	

1

2 **5. Mathematical Conventions**

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

7 **5.1 Mathematical Notation**

8 The following mathematical notation is used throughout the document.

9 0 denotes the integer 0, the bit 0, or the additive identity (the element zero) of a finite field. See Section 5.3
10 for more on finite fields.

- 1 1 denotes the integer 1, the bit 1, or the multiplicative identity (the element one) of a finite field. See
2 Section 5.3 for more on finite fields.
- 3 $a \times b$ denotes the product of a and b , where a and b are either both integers, or both finite field elements.
4 When it does not cause confusion, \times is omitted and the notation ab is used. See Section 5.3 for more on
5 finite fields.
- 6 $a \times P$ denotes scalar multiplication of an elliptic curve point P by a non-negative integer a . When it does
7 not cause confusion, \times is omitted and the notation aP is used. See Section 5.4 for more on elliptic curves.
- 8 $\lceil x \rceil$ denotes the smallest integer greater than or equal to the real number x . For example,
9 $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
- 10 $\lfloor x \rfloor$ denotes the largest integer less than or equal to the real number x . For example $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
- 11 $[a, b]$ denotes the interval of integers between and including the integers a and b .
- 12 $LCM(a, b)$ denotes the least common multiple of a and b for two positive integers a and b , i.e., the least
13 positive integer that is divisible by both a and b . See Annex A.1.1 and A.2.2 for an algorithm to compute
14 the LCM.
- 15 $GCD(a, b)$ denotes the greatest common divisor of a and b for two positive integers a and b , i.e., the largest
16 positive integer that divides both a and b . See Annex A.1.1 and A.2.2 for an algorithm to compute the
17 GCD.
- 18 $X \oplus Y$ denotes the bitwise exclusive-or (XOR) of two bit strings or two octet strings X and Y of the same
19 length.
- 20 $X \parallel Y$ denotes the ordered concatenation of two strings X and Y . X and Y are either both bit strings, or both
21 octet strings.
- 22 $\lg x$ denotes the logarithmic function of a positive number x to the base 2.
- 23 $\log_{256} x$ denotes the logarithmic function of a positive number x to the base 256.
- 24 $a \bmod n$ denotes the unique remainder r , $0 \leq r < n$, when the integer a is divided by the positive integer n .
25 For example, $23 \bmod 7 = 2$. The operator “mod” has the lowest precedence of all arithmetic operators
26 (e.g., $5 + 8 \bmod 3$ is equal to $13 \bmod 3$, not $5 + 2$). See Annex A.1.1 for more details.
- 27 $a \equiv b \pmod{n}$ denotes that $n \mid (a - b)$, so that the integers a and b have the same remainder when divided by
28 the positive integer n . Pronounced “ a is congruent to b modulo n .” This is equivalent to $(a \bmod n) = (b$
29 $\bmod n)$. See Annex A.1.1 for more details.
- 30 $a \not\equiv b \pmod{n}$ denotes that $n \nmid (a - b)$, so that the integers a and b have different remainders when divided
31 by the positive integer n . Pronounced “ a is not congruent to b modulo n .” This is equivalent
32 to $(a \bmod n) \neq (b \bmod n)$.

- 1 $a^{-1} \bmod n$ denotes a positive integer $b < n$, if it exists, such that $(ab) \bmod n = 1$. This is pronounced “the
 2 (multiplicative) inverse of a modulo n ,” and is also denoted by $\frac{1}{a} \bmod n$. See Annex A.1.1 and A.2.2 for a
 3 more detailed description and an algorithm for computing it.
- 4 $\frac{a}{b} \bmod n$ denotes an integer a multiplied by the inverse of the integer b , with the computations performed
 5 modulo n . Equivalent to $ab^{-1} \bmod n$.
- 6 $GF(p)$ denotes the finite field of p elements, represented as the integers modulo p , where p is an odd prime
 7 number. This is also known as a prime finite field. See Section 5.3.1 for more information.
- 8 $GF(p^m)$ denotes the finite field containing p^m elements for some integer $m > 1$, where p is an odd prime.
 9 If $n \mid m$ this is also known as an extension field of the field $GF(p^n)$.
- 10 $GF(q)$ denotes the finite field containing q elements. In the context of this document, q will be a power of
 11 an odd prime.
- 12 $E / GF(q)$ denotes an elliptic curve defined over the field $GF(q)$.
- 13 $E^t / GF(q)$ denotes the order twist of order t of the elliptic curve $E / GF(q)$.
- 14 $E(GF(q))$ denotes the additive group of points on the elliptic curve $E / GF(q)$.
- 15 $\#E(GF(q))$ denotes the order of the group $E(GF(q))$ or the number of points on an elliptic curve defined
 16 over the field $GF(q)$. This may be abbreviated to $\#E$ when the context is clear.
- 17 $e(P, Q)$ denotes a pairing, or an efficiently-computable, non-degenerate bilinear mapping. For an elliptic
 18 curve $E / GF(q)$ and an integer $n \mid \#E(GF(q))$ such a pairing takes as parameters elliptic curve points
 19 $P \in E(GF(q))[n]$ and $Q \in E(GF(p^k))$, and evaluates as an element in the multiplicative group $GF(p^k)^*$,
 20 where k is known as the *embedding degree*, or as the *security multiplier*.
- 21 $e_2(P, Q)$ denotes a pairing compressed by a factor of 2 and represented as an element in $GF(p^{k/2})$
- 22 $e_3(P, Q)$ denotes a pairing compressed by a factor of 3 and represented as an element in $GF(p^{k/3})$
- 23 $\phi(P)$ denotes a distortion map, or a non-rational endomorphism on an elliptic curve group $E(GF(q^2))$.
 24 Such a mapping maps a point $P \in E(GF(q))$ to a point $\phi(P) \in E(GF(q^2))$ such that P and $\phi(P)$ are linearly
 25 independent.
- 26 $\hat{e}(P, Q)$ denotes a modified pairing. See Section 5.4 for further details.
- 27 $\phi_d(P)$ denotes a mapping [isomorphism?] from $E^d(GF(q))$ to $E(GF(q^d))$

- 1 $\left(\frac{x}{n}\right)$ denotes a Jacobi symbol. See Annex A.1.4 for a detailed description and A.2.3 for an algorithm to
 2 compute Jacobi symbols.
- 3 \mathcal{O} denotes the point at infinity on an elliptic curve. See Section 5.4 for more information.
- 4 $\exp(a,b)$ denotes the result of raising a to the power b , where a is an integer or a finite field element, and b
 5 is an integer. This may also be denoted by a^b .
- 6 NOTE—Throughout this main document, integers and field elements are denoted with lower-case letters; while octet
 7 strings and elliptic curve points are denoted with upper-case letters.

8 5.2 Bit Strings and Octet Strings

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

14 NOTE—When a string is represented as a sequence, it may be indexed from right to left or from left to
 15 right, starting with any index; this does not change the meaning of the terms above. For example, consider
 16 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 $a_0 = 1c$, $a_1 = 76$,
 17 $a_2 = 3b$, and $a_3 = e4$. In this case, a_0 represents the first octet, and a_3 represents the last octet.
 18 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
 19 this case, a_1 represents the first octet, and a_4 represents the last octet. Yet another possibility would be to
 20 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
 21 octet and a_0 represents the last octet. No matter how this string is represented, the value of the first octet is
 22 always 1c and the value of the last octet is always e4.

23 5.3 Finite Fields

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

29 5.3.1 Prime Finite Fields

30 A prime finite field is a field containing a prime number of elements. If p is a prime, then there is a unique
 31 (up to isomorphism) field $GF(p)$ with p elements, and the elements of $GF(p)$ shall be represented by the
 32 integers $0, 1, 2, \dots, p - 1$.

33 A description of the arithmetic of $GF(p)$ is given in IEEE-1363 Annex A.1 and A.2.

1 5.3.2 Odd characteristic extension fields

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

$$8 \quad (a_{m-1}, a_{m-2}, \dots, a_2, a_1, a_0)$$

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

$$10 \quad a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_2x^2 + a_1x + a_0$$

11 This subfield in turn may be represented recursively in a similar way. At the bottom of this tower of
 12 extensions we eventually arrive at a vector of $GF(p)$ elements, where an element of $GF(p^k)$ is represented
 13 by the sum

$$14 \quad a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \dots + a_2x^2 + a_1x + a_0$$

15 where $0 \leq a_i \leq (p-1)$ for $0 \leq i \leq (k-1)$. The towering method used is represented in the form
 16 $a/b/c/\dots$ where a, b, c , etc., are integers whose product is k . For example, for $k = 12$, a
 17 $3/2/2$ representation means that an element in $GF(p^k)$ is represented as a cubic extension, over a
 18 quadratic extension, over another quadratic extension over the base field. The values a_i are then picked off
 19 from left to right as the values at the bottom of this tower, from most significant to least significant.
 20 Towering is not mandatory, but towering is strongly recommended.

21 The (small) constants required by the irreducible binomials that define the tower of extensions also form a
 22 part of the extension field representation.

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

24 5.3.3 Unitary extension fields

25 An extension field element that is unitary can be compressed, but this form of compression causes the loss
 26 of information. In a compressed form, however, such elements can still be accurately exponentiated using
 27 special algorithms. The result of pairing calculations of the types described in this standard are always
 28 unitary, and hence compression of the pairing value is always possible, and may be desirable. If the
 29 embedding degree is a multiple of 2 then compression by a factor 2 is possible, and the LUC method of
 30 exponentiation can be used. If the embedding degree is a multiple of 3, then compression by a factor of 3 is
 31 possible, and the XTR method of exponentiation can be used.

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

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

1 5.4 Pairings

2 The pairing-friendly elliptic curves for use with this standard will be either

- 3 — Supersingular curves with an embedding degree of $k = 2$
- 4 — Ordinary curves with an embedding degree of $k = 2^i 3^j$, where $i > 0$ and $j \geq 0$.

5 Details of recommended families of pairing-friendly curves can be found in Annex A.2.

6 Additive notation will always be used for elliptic curve points. The bilinear of a pairing will be described as

$$7 \quad e(aP, bQ) = e(P, Q)^{ab}$$

8 Supersingular curves always have a suitable distortion map $\phi: GF(q) \rightarrow GF(q^2)$. A modified pairing on a
 9 supersingular curve is denoted $\hat{e}(P, Q) = e(P, \phi(Q))$ where $e(P, Q)$ is the underlying pairing, which is
 10 always the Tate pairing in the case of supersingular curves. For the modified pairing $\hat{e}(P, Q)$ both
 11 parameters P and Q are elements of $E(GF(p))$.

12 For a supersingular curve of the form $E/GF(q): y^2 = x^3 + ax$ where $q \equiv 3 \pmod{4}$, a distortion map is of the
 13 form

$$14 \quad \phi(x, y) = (-x, iy)$$

15 For a supersingular curve of the form $E/GF(q): y^2 = x^3 + b$ where $q \equiv 2 \pmod{3}$, a distortion map is of the
 16 form

$$17 \quad \phi(x, y) = (\xi x, y)$$

18 where $\xi^3 = 1$ and $\xi \neq 1$.

19 The distortion map ϕ must be specified as part of the specification of the modified
 20 pairing $\hat{e}(P, Q) = e(P, \phi(Q))$.

21 A modified pairing on an ordinary curve is denoted $\hat{e}(P, Q) = e(P, \phi_d(Q))$ where $e(P, Q)$ is the underlying
 22 pairing and $\phi_d: E^d(GF(q^{k/d})) \rightarrow GF(q^k)$ is one of the isomorphisms defined below, d is 2, 4 or 6,
 23 and $d \mid k$.

24 For an ordinary curve $E/GF(q^d)$ which has a twist of degree 2, the mapping ϕ_2 is of the form

$$25 \quad \phi_2(x, y) = (v^{-1}x, v^{-3/2}y)$$

26 where $v \in GF(q^d)$ is a quadratic non-residue in $GF(q^d)$.

27 For an ordinary curve $E/GF(q^d)$ which has a twist of degree 3, the mapping ϕ_3 is of the form

$$28 \quad \phi_3(x, y) = (v^{-1/3}x, v^{-1/2}y)$$

1 where $v \in GF(q^d)$ does not have a cube root in $GF(q^d)$.

2 For an ordinary curve $E/ GF(q^d)$ which has a twist of degree 6, the mapping ϕ_6 is of the form

$$3 \quad \phi_6(x, y) = (v^{-1/3}x, v^{-1/2}y)$$

4 where $v \in GF(q^d)$ does not have a sixth root in $GF(q^d)$.

5 The mapping ϕ_d must be specified as part of the specification of the modified pairing $\hat{e}(P, Q) = e(P, \phi_d(Q))$.

6 The first parameter to any pairing will always be a point of prime order p . The pairing evaluates as a unique
7 element in $GF(q^k)^*$ of order p (which may optionally be represented in compressed form, compressed by a
8 factor of either 2 or 3).

9 **5.5 Elliptic Curves and Points**

10 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
11 of $GF(q)$ that satisfy a certain equation, together with the point at infinity denoted by O . For the purposes
12 of this standard, it is specified by two field elements $a \in GF(q)$ and $b \in GF(q)$, called the *coefficients* of E .
13 The field elements x_p and y_p are called the x -coordinate of P and the y -coordinate of P , respectively.

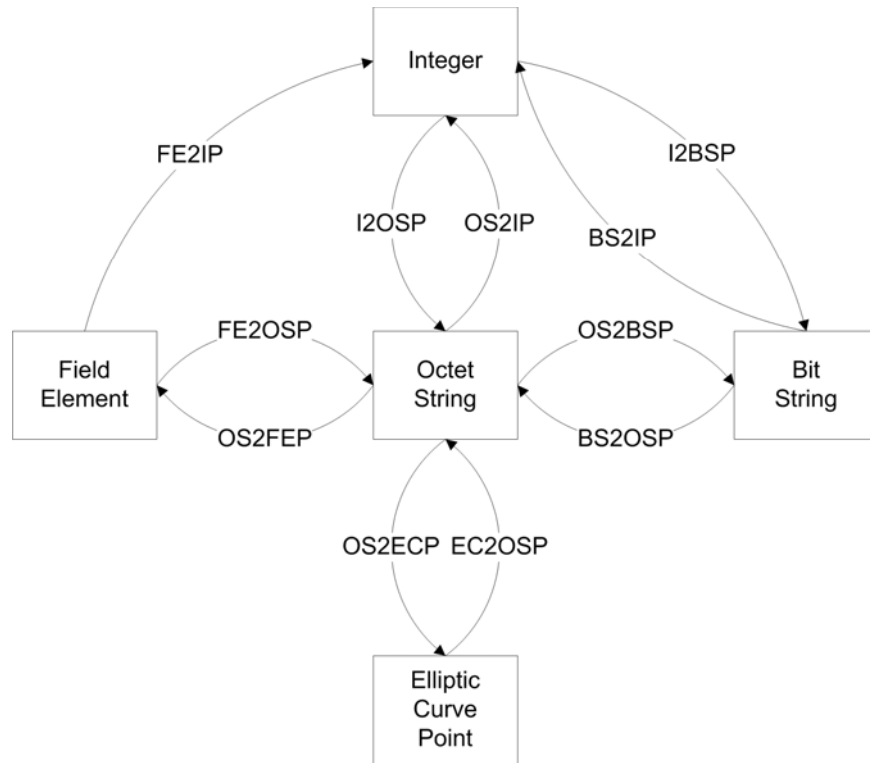
14 If $q = p^m$, $p > 3$ and $m \geq 1$, then a and b shall satisfy $4a^3 + 27b^2 \neq 0$ in $GF(q)$ and every point
15 $P = (x_p, y_p)$ in $E(GF(q))$ other than the point O shall satisfy the equation $y_p^2 = x_p^3 + a \cdot x_p + b$ in $GF(q)$.

16 See IEEE-1363 Annex A.9 and A.10 for more on elliptic curves and elliptic curve arithmetic.

17 **5.6 Data Type Conversion**

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

23 **Figure 1 Data type conversion primitives**



1

2 **5.6.1 Converting Between Integers and Bit Strings: I2BSP and BS2IP**

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

5 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
6 shall be written in its unique l -digit representation base 2:

7
$$x = x_{l-1}2^{l-1} + x_{l-2}2^{l-2} + \dots + x_1 2 + x_0$$

8 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
9 b_i have the value x_{l-i} for $1 \leq i \leq l$. The bit string shall be $b_1 b_2 \dots b_l$.

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

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

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

17 **5.6.2 Converting Between Bit Strings and Octet Strings: BS2OSP and OS2BSP**

18 To represent a bit string as an octet string, one simply pads enough zeroes on the left to make the number of
19 bits a multiple of 8, and then breaks it up into octets. More precisely, a bit string $b_{l-1}b_{l-2} \dots b_1 b_0$ of length l

1 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 $0 \leq i < (d-1)$,
 2 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
 3 and its rightmost $8-(8d-l)$ bits shall be $b_{l-1}b_{l-2}\dots b_{8d-8}$.

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

6 The primitive that converts octet strings to bit strings is called Octet String to Bit String Conversion
 7 Primitive or OS2BSP. It takes an octet string of length d and the desired length l of the bit string as input.
 8 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
 9 output “error” otherwise.

10 5.6.3 Converting between Integers and Octet Strings: I2OSP and OS2IP

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

$$13 \quad x = x_{l-1}256^{l-1} + x_{l-2}256^{l-2} + \dots + x_1256 + x_0$$

14 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
 15 have the value x_i for $0 \leq i < (l-1)$. The octet string shall be $M_{l-1}M_{l-2}\dots M_1M_0$.

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

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

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

23 5.6.4 Converting between finite field elements and octet strings: FE2OSP and OS2FEP

24 An element x of a finite field $GF(q)$, for purposes of conversion, is represented by an integer if q is an odd
 25 prime or an odd prime power (see 5.3.1 and 5.3.2). If q is an odd prime or an odd prime power, then to
 26 represent x as an octet string, I2OSP shall be used with the integer value representing x and the length
 27 $\lceil \log_{256} q \rceil$ as inputs.

28 The primitive that converts finite field elements to octet strings is called the Field Element to Octet String
 29 Conversion Primitive or FE2OSP. It takes a field element x , the field size q , and both the field characteristic
 30 p and the extension degree k if q is an odd prime-power as inputs, and outputs the corresponding octet
 31 string.

32 To convert an octet string back to a field element, if q is an odd prime or an odd prime power, then OS2IP
 33 shall be used with the octet string as the input. The primitive that converts octet strings to finite field
 34 elements is called the Octet String to Field Element Conversion Primitive or OS2FEP. It takes the octet
 35 string and the field size q as inputs and outputs the corresponding field element. It shall output “error” if
 36 OS2BSP or OS2IP outputs “error.”

1 **5.6.5 Converting Finite Field Elements to Integers: FE2IP**

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

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

- 7 1) Convert the element α to an octet string using FE2OSP.
- 8 2) Convert the resulting octet string to an integer i using OS2IP.
- 9 3) Output i .

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

13 **5.6.6 Converting between elliptic curve points and octet strings**

14 An elliptic curve point P (which is not the point at infinity \mathcal{O}) can be represented in either compressed or
15 uncompressed form. (For internal calculations, it may be advantageous to use other representations; e.g.,
16 the projective coordinates of A.9.6. Also see A.9.6 also for more information on point compression.) The
17 uncompressed form of P is simply given by its two coordinates. The compressed form is presented below.
18 The octet string format is defined to support both compressed and uncompressed points.

19 **5.6.6.1 Compressed elliptic curve points**

20 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
21 the x -coordinate of P , and y_p is a bit which is computed as defined below.

22 Two compressed forms are defined: an *LSB compressed form* is defined for elliptic curves over $GF(p)$ and
23 an *SORT compressed form* is defined for elliptic curves over $GF(p^m)$.

24 **5.6.6.1.1 LSB compressed form**

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

27 Procedures for *point decompression* (i.e., recovering \hat{y}_p given x_p and y_p) are given in A.12.8.

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

30 **5.6.6.1.2 SORT compressed form**

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

- 1 The SORT compressed form has $\tilde{y}_p = 1$ if $FE2IP(y_p) > FE2IP(y'_p)$ and $\tilde{y}_p = 0$ otherwise.
- 2 A procedure for point decompression is given in A.12.11.
- 3 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
4 first computing $FE2IP(y_p)$ and $FE2IP(y'_p)$.
- 5 NOTE 2—Although the SORT compressed form is defined here for any field, in the representations below it is only
6 employed for elliptic curves over $GF(p^m)$.
- 7 NOTE 3—The name “SORT” refers to the fact that the compressed bit is based on comparing the integer
8 representations of y_p and y'_p .

9 5.6.6.2 Two-coordinate point representations

10 For all the conversion primitives in this section, the point \mathcal{O} shall be represented by an octet string
11 containing a single 0 octet. The rest of this section discusses octet string representation of a point $P \neq \mathcal{O}$.
12 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
13 representation of P in one of the forms above.

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

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

$$19 \quad PO = PC \parallel X \parallel Y$$

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

- 21 — Bit S is 1 if the format uses the SORT compressed form; 0 otherwise.
- 22 — Bit U is 1 if the format is uncompressed or hybrid; 0 otherwise.
- 23 — Bit C is 1 if the format is compressed or hybrid; 0 otherwise.
- 24 — Bit \tilde{Y} is equal to the bit \tilde{y}_p if the format is compressed or hybrid; 0 otherwise.
- 25 — X is the octet string of length $\lceil \log_{256} q \rceil$ representing x_p according to FE2OSP (see 5.6.4).
- 26 — 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
27 format is uncompressed or hybrid; Y is an empty string if the format is compressed.

28 The primitive that converts elliptic curve points to octet strings for a given representation is called Elliptic
29 Curve Point to Octet String Conversion Primitive–R, or EC2OSP–R, where R is the representation. It takes
30 an elliptic curve point P and the size q of the underlying field as input and outputs the corresponding octet
31 string PO .

32 The primitive that converts octet strings to elliptic curve points is called Octet String to Elliptic Curve Point
33 Conversion Primitive–R, or OS2ECP–R. It takes the octet string and the field size q as inputs and outputs
34 the corresponding elliptic curve point, or “error.” It shall use OS2FEP to get x_p . It shall use OS2FEP to

1 get y_p if the format is uncompressed, and may output “error” if the recovered point is not on the elliptic
 2 curve. It shall use point decompression (see 5.5.6.1) to get y_p if the format is compressed. It can get y_p by
 3 either of these two means if the format is hybrid, and, if the format is hybrid, may output “error” if different
 4 values are obtained by the two means. It shall output “error” in the following cases:

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

9 The pairs of primitives for each of five representations are defined in the following sections.

10 **5.6.6.2.1 Uncompressed representation: EC2OSP-XY and OS2ECP-XY**

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

12 In this representation, the octet PC shall have binary value 0000 0100 and the octet strings X and Y shall
 13 represent x_p and y_p respectively. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The
 14 corresponding primitives are called EC2OSP-XY and OS2ECP-XY.

15 **5.6.6.2.2 LSB compressed representation: EC2OSP-XL and OS2ECP-XL**

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

17 In this representation, the octet PC shall have binary value 0000 001 \tilde{Y} where \tilde{Y} is equal to the bit \tilde{y}_p in the
 18 LSB compressed form, the octet string X shall represent x_p , and the octet string Y shall be the empty string.
 19 The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called
 20 EC2OSP-XL and OS2ECP-XL.

21 **5.6.6.2.3 SORT compressed representation: EC2OSP-XS and OS2ECP-XS**

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

23 In this representation, the octet PC shall have binary value 0000 101 \tilde{Y} where \tilde{Y} is equal to the bit y_p in the
 24 SORT compressed form, the octet string X shall represent x_p , and the octet string Y shall be the empty
 25 string. The length of the octet string PO shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called
 26 EC2OSP-XS and OS2ECP-XS.

27 **5.6.6.2.4 LSB hybrid representation: EC2OSP-XYL and OS2ECP-XYL**

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

1 In this representation, the octet *PC* shall have binary value 0000 011 \tilde{Y} where \tilde{Y} is equal to the bit y_p in the
 2 LSB compressed form, and the octet strings *X* and *Y* shall represent x_p and y_p respectively. The length of
 3 the octet string *PO* shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSP-XYL and
 4 OS2ECP-XYLSORT hybrid representation: EC2OSP-XYS and OS2ECP-XYS

5 **5.6.6.2.5 SORT hybrid representation: EC2OSP-XYS and OS2ECP-XYS**

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

7 In this representation, the octet *PC* shall have binary value 0000 111 \tilde{Y} where \tilde{Y} is equal to the bit \tilde{y}_p in the
 8 SORT compressed form, and the octet strings *X* and *Y* shall represent x_p and y_p respectively. The length
 9 of the octet string *PO* shall be $1 + 2 \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSP-XYS and
 10 OS2ECP-XYS.

11 **5.6.6.3 x-coordinate only representation: EC2OSP-X and OS2ECP-X**

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

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

15 In this representation, the octet *PC* shall have binary value 0000 0001, the octet string *X* shall represent x_p ,
 16 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
 17 corresponding primitives are called EC2OSP-X and OSC2ECP-X.

18 OS2ECP-X may output any of the (at most two) elliptic curve points with the given *x*-coordinate. Thus, the
 19 original *y*-coordinate is not necessarily recovered.

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

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

26 **5.6.6.4 Summary of representations**

27 Table 2 summarizes the point representations in 5.5.6.2 and 5.5.6.3.

28 **Table 2 Elliptic curve point representations**

Representation	Primitives	<i>PC</i>	<i>X</i>	<i>Y</i>	Finite fields
Uncompressed	EC2OSP-XY OS2ECP-XY	0000 0100	x_p	y_p	All

LSB compressed	EC2OSP-XL OS2ECP-XL	0000 001 \tilde{Y}	x_p	Empty	$GF(p)$
SORT compressed	EC2OSP-XS OS2ECP-XS	0000 101 \tilde{Y}	x_p	Empty	$GF(p^m)$
LSB hybrid	EC2OSP-XYL OS2ECP-XYL	0000 011 \tilde{Y}	x_p	y_p	$GF(p)$
SORT hybrid	EC2OSP-XYS OS2ECP-XYS	0000 111 \tilde{Y}	x_p	y_p	$GF(p^m)$
x -coordinate-only	EC2OSP-X OS2ECP-X	0000 0001	x_p	Empty	All
Point \mathcal{O}	All	0000 0000	Empty	Empty	All

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

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

8 6. Hashing Primitives

9 The schemes described in Sections 7.2 through 7.4 require the use of two or more cryptographic hash
10 functions. Implementation of the schemes described in these sections must define the Each of these hash
11 functions is based on one of the hash functions defined by NIST in FIPS 180-1, *Secure Hash Standard*,
12 which will be denoted H . The hash function H chosen for will depend on a security parameter which
13 defines the level of bit strength that is required. This bit strength is required to be one of the standard levels:
14 80, 112, 128, 192 or 256. The hash function H is then used to construct the required hash function as
15 described below.

16 6.1 Hashing to an integer

17 The function IHF1 is a cryptographic hash function that hashes a string to an integer. The function IHF1-
18 SHA uses the SHA-1 and SHA-2 family [REF] to accomplish this. Other hash functions can be constructed
19 as needed.

20 6.1.1 The Function IHF1-SHA

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

23 Input:

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

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

3 Output:

4 — An integer $v \in Z_n$.

5 Operation: Use the following steps.

- 6 1) If $t = 80$ then let $H = \text{SHA} - 1$
- 7 2) else if $t = 112$ then let $H = \text{SHA} - 224$
- 8 3) else if $t = 128$ then let $H = \text{SHA} - 256$
- 9 4) else if $t = 192$ then let $H = \text{SHA} - 384$
- 10 5) else if $t = 256$ then let $H = \text{SHA} - 512$
- 11 6) Let $v_0 = 0$
- 12 7) Let $h_0 = 0x00\dots00$, string of null bytes of length $2t$
- 13 8) Let $t_1 = h_0 \parallel s$
- 14 9) Let $h_1 = H(t_1)$
- 15 10) Let $a_1 = \text{I2OSP}(h_1)$
- 16 11) Let $v_1 = a_1$
- 17 12) Let $t_2 = h_1 \parallel s$
- 18 13) Let $a_2 = \text{I2OSP}(h_2)$
- 19 14) Let $v_2 = 256^{2^x} v_1 + a_2$
- 20 15) Output $v_2 \bmod n$

21 6.2 Hashing to a string

22 The function IHF1 is a cryptographic hash function that hashes a string to a string. The function SHF1-
 23 SHA uses the SHA-1 and SHA-2 family [REF] to accomplish this. Other hash functions can be constructed
 24 as needed.

25 6.2.1 The Function SHF1-SHA

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

27 Input:

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

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

3 Output:

4 — A string $v \in \{0,1\}^n$

5 Operation: Use the following steps.

6 1) Output $IHF1(s, 2^n, t)$.

7 **6.3 Hashing to a point on a curve**

8 The function PHF1 is a cryptographic hash function that hashes a string to a point on a supersingular
 9 elliptic curve. The function PHF1-SHA uses the SHA-1 and SHA-2 family [REF] to accomplish this. Other
 10 hash functions can be constructed as needed.

11 **6.3.1 The Function PHF1-SHA**

12 Returns an element of an elliptic curve group $E(GF(q))[p]$ for a supersingular elliptic curve

13 $E/GF(q): y^2 = x^3 + 1$ or $E/GF(q): y^2 = x^3 + x$.

14 Input:

15 — A string $s \in \{0,1\}^*$

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

17 — A flag j taking the values 0 or 1 which defines a supersingular elliptic curve, with $j = 0$ representing
 18 the elliptic curve and representing the elliptic curve $E/GF(q): y^2 = x^3 + 1$ and $j = 1$ representing the
 19 elliptic curve $E/GF(q): y^2 = x^3 + x$.

20 — A prime q with $q \equiv 11 \pmod{12}$ that defines the finite field $GF(q)$.

21 — 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 .

22 Assumptions: TBD

23 Output:

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

25 Operation: Use the following steps.

26 1) Let $r = (q+1)/p$.

27 2) If $j = 0$ then perform the following steps:

28 i) Let $y = IHF1(s, q, t)$.

29 ii) Let $x = (y^2 - 1)^{(2q-1)/3} \pmod{q}$.

- 1 iii) Let $Q = (x, y)$.
- 2 3) Else if $j = 1$ perform the following steps:
- 3 i) Let $x = IHFI(s, q, t)$.
- 4 ii) If the Jacobi symbol $(x/q) = +1$ then perform the following steps
- 5 i) Let $y = x^{(q+1)/4} \bmod q$.
- 6 ii) Let $Q = (x, y)$.
- 7 iii) Else perform the following steps:
- 8 i) Let $y = (-x)^{(q+1)/4} \bmod q$.
- 9 ii) Let $Q = (-x, y)$.
- 10 4) Return rQ .

11 7. Pairing-based Primitives

12 This section describes the mathematical operations that are used to build the pairing-based algorithms
 13 described in Section 7. The primitives are divided into four families, each based on a different security
 14 assumption. Each family defines a generation, verification, encryption, and decryption operation. Note
 15 that the operations must be used as a set, and cannot be intermixed.

16 The operations are defined in a setting containing a sender, a receiver, and a key server. The key server
 17 possesses some secret information s , and publishes a set of public parameters \mathcal{P} . The receiver is identified
 18 by a bit string ID . The four operations perform the following actions in this setting:

- 19 — **Generation:** This operation is performed by the key server. It combines the string ID with the
 20 secret information s to derive the receiver's private key K_{ID} .
- 21 — **Verification:** This operation is performed by the receiver. It combines K_{ID} , ID , \mathcal{P} , and, with some
 22 primitives, the encrypted message c . It outputs "valid" if K_{ID} corresponds to ID and \mathcal{P} . It outputs
 23 "invalid" otherwise. This operation is optional to the operation of most of the schemes.
- 24 — **Encryption:** The operation is performed by the sender. It combines \mathcal{P} , a plaintext message m , and
 25 ID to construct the ciphertext c . Note that "encryption" as defined here is only the basis for an
 26 actual key encapsulation or encryption scheme defined later.
- 27 — **Decryption:** This operation is performed by the receiver. It combines \mathcal{P} , and the ciphertext c to
 28 compute the plaintext m . As with encryption, this decryption operation is the basis for an
 29 implemented key decapsulation or decryption operation defined by the scheme.

30 In all the primitives below, the following elements are required to be defined:

- 31 — G_1 , a group of prime order p
- 32 — G_2 , a group of prime order p
- 33 — G_3 , a group of prime order p
- 34 — A pairing $e : G_1 \times G_2 \rightarrow G_3$

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

4 **7.1 Pairing-based SK primitives**

5 These primitives define operations with cryptographic strength defined by reduction to the q -Bilinear
 6 Diffie-Hellman Inverse (q -BBDHI) problem, defined as:

7 Given group elements $g_1, g_2, g_2^x, g_2^{x^2}, \dots, g_2^{x^q}$, compute $e(g_1, g_2)^{1/x}$

8 For more information on this problem, the proof that reduces the problem of breaking these primitives to q -
 9 BDHI, and information on the associated primitives, see [Chen2005].

10 For all these operations, the systems parameters are assumed to be defined as:

11 The server secret is defined to be:

12 — s , a random element of Z_p .

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

14 — G_1, G_2, G_3, e , the system parameters as defined above in Section 6

15 — Q_1 : a generator of G_1

16 — Q_2 : a generator of G_2 , defined such that $Q_1 = \phi(Q_2)$

17 — R : sQ_1

18 **7.1.1 Pairing-based SK: Generation (P-SK-G)**

19 Input:

20 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R)$

21 — The server secret s

22 — An encoded identity $M \in Z_p$, typically derived from an identity string

23 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; s is the private key corresponding to R , so
 24 that $R = sQ_1$; and M is an element of Z_p .

25 Output:

26 — The derived private key K_M , an element of G_2 , or “error”

27 Operation: Use the following steps to compute K_M .

28 1) If $M + s \equiv 0 \pmod{p}$, output "error" and stop.

29 2) Compute $i = (M + s)^{-1} \pmod{p}$.

1 3) Compute $K_M = iQ_2$.

2 4) Output K_M .

3 **7.1.2 Pairing-based SK: Verification (P-SK-V)**

4 Input:

5 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R)$

6 — A public key element M in Z_p

7 — The corresponding private key K_M .

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

10 Output:

11 — The value "valid" if is consistent with R and M and "invalid" otherwise

12 Operation: Use the following steps.

13 1) Compute $T_1 = e(K_M, MQ_2 + R)$

14 2) Compute $T_2 = e(Q_1, Q_2)$

15 3) If $T_1 = T_2$ then output the value "valid", otherwise output the value "invalid."

16 NOTE—The value $T_2 = e(Q_1, Q_2)$ is a constant, and can be precomputed once and cached, saving a pairing operation.

17 **7.1.3 Pairing-based SK: Encryption (P-SK-E)**

18 Input:

19 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R)$

20 — The public key element M

21 — A message randomizer r , an integer

22 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is the public parameter associated with the
23 server secret s ; M is an element of Z_p .

24 Output:

25 — A ciphertext E , where E is an element of , along with a blinding factor B , an element of G_1 , along
26 with a blinding factor B , an element of G_3

27 Operation: Use the following steps.

28 1) Compute $E = r(MQ_1 + R)$.

29 2) Compute $B = e(p_1, p_2)^r$.

1 3) Output E and B .

2 NOTE—The blinding factor B is the same as that obtained in P-SK-D, below.

3 **7.1.4 Pairing-based SK: Decryption (P-SK-D)**

4 Input:

5 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R)$

6 — The public key element M

7 — The associated private key K_M

8 — A ciphertext E

9 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is a public key, an element of G ; M is an
10 element of Z_p ; K_{ID} , an element of G , is a valid generated value for M according to P-SK-GV; and E is an
11 element of G_1 .

12 Output:

13 — A blinding factor B , an element of G_3 .

14 Operation: Use the following steps.

15 1) Compute $B = e(E, K_M)$.

16 2) Output B .

17 NOTE—If both parties follow the algorithms specified, the value B computed here is $e(p_1, p_2)^r$, the same as that
18 obtained in P-SK-E, above.

19 **7.2 Pairing-based BB₁ primitives**

20 These primitives define operations with cryptographic strength defined by reduction to the Bilinear Diffie-
21 Hellman (BDH) problem, defined as:

22 Given group elements $g_1, g_2, g_1^x, g_1^y, g_2^x, g_2^z$, compute $e(g_1, g_2)^{xyz}$

23 For more information on this problem, the proof that reduces the problem of breaking these primitives to
24 BDH, and information on the associated primitives, see [BB2004].

25 For all these operations, the systems parameters are assumed to be defined as:

26 The server secret that is defined to be:

27 — s , comprising three random elements s_1, s_2, s_3 , of Z_p

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

29 — G_1, G_2, G_3, e : the system parameters as defined above in Section 6

- 1 — Q_1 : a generator of G_1
 2 — Q_2 : a generator of G_2
 3 — $R = s_1 Q_1$
 4 — $T = s_3 Q_1$
 5 — $V = e(R, s_2 Q_2)$

6 NOTE—There exists an isomorphism $\phi : G_2 \rightarrow G_1$ such that $Q_1 = \phi(Q_2)$ but the commutative blinding primitives do
 7 not require its explicit knowledge.

8 **7.2.1 Pairing-based BB₁: Generation (P-BB1-G)**

9 Input:

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

14 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; s is the server secret corresponding to \mathcal{P} ; and
 15 M is an element of Z_p .

16 Output:

- 17 — The derived secret key, $K_{0,M}$ and $K_{1,M}$, two elements of G_2

18 Operation: Use the following steps to compute $K_{0,M}$ and $K_{1,M}$.

- 19 1) Compute $i = s_1 s_2 + r(s_1 M + s_3)$ in Z_p .
 20 2) Compute $K_{0,M} = i Q_2$.
 21 3) Compute $K_{1,M} = r Q_2$.
 22 4) Output $K_{0,M}$ and $K_{1,M}$.

23 **7.2.2 Pairing-based BB₁: Verification (P-BB1-V)**

24 Input:

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

28 Assumptions: the parameters \mathcal{P} describe valid bilinear groups and correspond to the server secret s ; M is an
 29 element of Z_p ; and $K_{0,M}$ and $K_{1,M}$ are two elements of G_2 .

1 Output:

2 — The value "valid" if $K_{0,M}$ and $K_{1,M}$ are consistent with P and M and "invalid" otherwise

3 Operation: Use the following steps.

4 1) Compute $T_0 = e(Q_1, K_{0,M})$.

5 2) Compute $T_1 = e(MR + T, K_{1,M})$.

6 3) If $T_0 = T_1V$ then output the value "valid," otherwise output the value "invalid."

7 7.2.3 Pairing-based BB₁: Encryption (P-BB1-E)

8 Input:

9 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R, T, V)$

10 — The public key element M in Z_p .

11 — A message randomizer r , an element of Z_p .

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

14 Output:

15 — An ciphertext E_0 and E_1 , both elements of G_1 , along with a blinding factor B , an element of G_3

16 Operation: Use the following steps.

17 1) Compute $E_0 = rQ_1$.

18 2) Compute $E_1 = (rM)R + rT$.

19 3) Compute $B = V^r$.

20 4) Output E_0, E_1 , and B .

21 NOTE—The point multiplications in steps 1 and 2 and the field exponentiation in step 3 can be performed very
22 efficiently using classic pre-computation algorithms. Pre-computation is viable and inexpensive because the base
23 elements Q_1, R, T , and V , are all part of the public parameters and do not depend on the recipient identity.

24 7.2.4 Pairing-based BB₁: Decryption (P-BB1-D)

25 Input:

26 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q_1, Q_2, R, T, V)$

27 — The public key element M

28 — The associated private key $K_{0,M}$ and $K_{1,M}$

29 — A ciphertext E_0 and E_1

1 Assumptions: The parameters \mathcal{P} describe valid bilinear groups and public parameters; M is an element
 2 of Z_p ; $K_{0,M}$ and $K_{1,M}$ are elements of G_2 corresponding to M according to P-BB1-V; and E_0 and E_1 are
 3 elements of G_1 .

4 Output:

5 — A blinding factor B , an element of G_3

6 Operation: Use the following steps.

7 1) Compute $B = e(E_0, K_{0,M}) / e(E_1, K_{1,M})$.

8 2) Output B .

9 NOTE—If both parties follow the algorithms specified, the value B computed here is V' , the same as that obtained in
 10 PBB1E, above.

11 7.3 Pairing-based BF primitives

12 These primitives define operations with cryptographic strength defined by reduction to the Bilinear Diffie-
 13 Hellman problem, defined as:

14 Given group elements g_1, g_2^x, g_2^y, g_2^z , compute $e(g_1, g_2)^{xyz}$

15 For more information on this assumption, and the proof that these operations reduce to this problem, see
 16 [Boneh2001.] This system differs from the primitives above in that the identity is encoded into an element
 17 of G_1 , instead of into Z_p . The mapping from a string to an element of G_1 , which is typically an elliptic
 18 curve group, is more complex than mapping onto an integer in Z_p , which is where these primitives get their
 19 name.

20 For all these operations, the systems parameters are assumed to be defined as:

21 The server secret is defined to be

22 — s , a random element of Z_p

23 The public parameters are defined to be $\mathcal{P} = (G_1, G_2, G_3, e, Q, R)$ where

24 — G_1, G_2, G_3, e , the system parameters as defined above in Section 6

25 — Q , a generator of G_2

26 — R , equal to sQ , an element of G_2

27 7.3.1 Pairing-based BF: Generation (P-BF-G)

28 Input:

29 — The parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q, R)$

30 — The server secret s

1 — An encoded identity M in G_1 , typically derived from an identity string
 2 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; s is the server secret corresponding to R ; and
 3 M is an element of G_1 .

4 Output:

5 — The derived secret D , an element of G_1

6 Operation: Use the following steps.

7 1) Compute $D = sM$.

8 2) Output D .

9 NOTE—The details of encoding identities into values M vary depending on the construction that employs the P-FDH
 10 primitives. Not all possible encoding methods will yield secure schemes. Some constructions will include additional
 11 elements in the FDH public key to be used in computing M .

12 7.3.2 Pairing-based BF: Verification (P-BF-V)

13 Input:

14 — The public parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q, R)$

15 — An encoded identity M , an element of G_1

16 — The purported generated key D

17 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; D is an FDH public key, and thus an element
 18 of G_1 ; M is an element G_1 ; and R is an element of G_1 .

19 Output:

20 — The value "valid" if D is consistent with M and R and "invalid" otherwise

21 Operation: Use the following steps.

22 1) Compute $T_1 = e(D, Q)$.

23 2) Compute $T_2 = e(M, R)$.

24 3) If $T_1 = T_2$ then output the value "valid," otherwise output the value "invalid."

25 7.3.3 Pairing-based BF: Encryption (P-BF-E)

26 Input:

27 — The public parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q, R)$

28 — An encoded identity M , an element of G_1

29 — A per-message random integer r

30 Assumptions: the parameters \mathcal{P} describe valid bilinear groups.

1 Output:

2 — A ciphertext E , an element of G_2 , along with a blinding factor B , an element of G_3 .

3 Operation: Use the following steps.

4 1) Compute $E = rQ$.

5 2) Compute $B = e(rM, R)$.

6 3) Output E and B .

7 NOTE—The blinding factor B is the same as that obtained in P-BF-D, below.

8 **7.3.4 Pairing-based BF: Decryption (P-BF-D)**

9 Input:

10 — The public parameters $\mathcal{P} = (G_1, G_2, G_3, e, Q, R)$

11 — A ciphertext C

12 — A generated value D

13 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; C is an element of G_1 ; D , an element of G_2 ,
14 is a valid generated value according to P-BF-G.

15 Output:

16 — A blinding factor B , an element of G_3

17 Operation: Use the following steps.

18 1) Compute $B = e(C, D)$.

19 2) Output B .

20 NOTE—If both parties follow the algorithms specified, the value B computed here is $e(U, X)^s$, the same blinding
21 factor obtained in P-BF-E, above

22 **7.4 Pairing-based HIBE primitives**

23 These primitives define operations with cryptographic strength defined by reduction to the Bilinear Diffie-
24 Hellman (BDH) problem, defined as:

25 Given group elements g_1, g_1^x, g_1^y, g_1^s , compute $\hat{e}(g_1, g_1)^{xys}$

26 For more information on this problem, the proof that reduces the problem of breaking these primitives to
27 BDH, and information on the associated primitives, see [BB2004].

28 For all these operations, the systems parameters are assumed to be defined as:

29 The server secrets are defined to be:

- 1 — s_0, s_1, \dots, s_n , random elements of Z_p
- 2 — The derived secret keys S_0, S_1, \dots, S_n , elements of G_2 , where $S_0 = O$
- 3 The public parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0)$ are defined to be:
- 4 — G_1, G_2, G_3, e , the system parameters as defined above in Section 6
- 5 — P_0 , a generator of G_2
- 6 — $Q_0 = s_0 P_0$
- 7 — The auxiliary public parameters Q_1, Q_2, \dots, Q_n , elements of G_2
- 8 The identity strings are defined to be:
- 9 — ID_0, ID_1, \dots, ID_n , strings

10 **7.4.1 Pairing-based HIBE: Generation (P-HIBE-G)**

11 Input:

- 12 — The parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0)$
- 13 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-2}
- 14 — Server secret s_{t-1} and a derived secret key S_{t-1}
- 15 — Encoded identity P_t in G_2 , where P_t is typically derived from the identity string ID_0, ID_1, \dots, ID_t

16 Assumptions: The parameters \mathcal{P} describe valid bilinear groups.

17 Output:

- 18 — Derived secret key S_t , element of G_2
- 19 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} , elements of G_2

20 Operation: Use the following steps to compute S_t and Q_{t-1} .

- 21 1) Compute $S_t = S_{t-1} + s_{t-1} P_t$
- 22 2) Compute $Q_{t-1} = s_{t-1} P_0$
- 23 3) Output S_t and Q_1, Q_2, \dots, Q_{t-1}

24 **7.4.2 Pairing-based HIBE: Verification (P-HIBE-V)**

25 Input:

- 26 — Parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0)$
- 27 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} in G_2

- 1 – Derived secret key S_{t-1}
 2 – Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is typically derived from the identity
 3 string ID_0, ID_1, \dots, ID_i
 4
 5 Output:
 6 – The value "valid" if secret key S_i and auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} are consistent
 7 with P and P_i or "invalid" otherwise

8 Operation: Use the following steps.

- 9 1) Compute $T = \hat{e}(Q_0, S_t)$
 10 2) Compute $V = \prod_{i=1}^t \hat{e}(Q_{i-1}, P_i)$
 11 3) If $T = V$ then output the value "valid," otherwise output the value "invalid."

12 7.4.3 Pairing-based HIBE: Encryption (P-HIBE-E)

13 Input:

- 14 – Parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0)$
 15 – Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is typically derived from the identity
 16 string ID_0, ID_1, \dots, ID_i
 17 – Message randomizer r , an element of Z_p .

18 Output:

- 19 – Ciphertext $U_0, U_2, U_3, \dots, U_t$, elements of G_2 ,
 20 – Blinding factor B , element of G_3

21 Operation: Use the following steps.

- 22 1) Compute $U_0 = rP_0$
 23 2) Compute $U_i = rP_i$ for $2 \leq i < t$
 24 3) Compute $B = \hat{e}(Q_0, P_1)^r$
 25 4) Output U_0 and U_2, U_3, \dots, U_t and B .

26 NOTE—The point multiplications in steps 1 and 2 and the field exponentiation in step 3 can be performed
 27 very efficiently using classic pre-computation algorithms. Pre-computation is viable and inexpensive
 28 because the base elements Q_1, R, T , and V , are all part of the public parameters and do not depend on the
 29 recipient identity.

30 7.4.4 Pairing-based HIBE: Decryption (P-HIBE-D)

31 Input:

- 1 — The parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0)$
- 2 — Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is typically derived from the identity string
- 3 ID_0, ID_1, \dots, ID_i
- 4 — Derived secret key S_i and auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1}
- 5 — A ciphertext U_0, U_2, \dots, U_t

6 Assumptions: The parameters \mathcal{P} describe valid bilinear groups and public parameters; S_i and

7 Q_1, Q_2, \dots, Q_{t-1} are elements of G_2 ; and U_0, U_2, \dots, U_t are elements of G_2

8 Output:

- 9 — Blinding factor B , element of G_3

10 Operation: Use the following steps.

11 1) Compute $B = \frac{\hat{e}(U_0, S_t)}{\prod_{i=2}^t \hat{e}(Q_{i-1}, U_i)}$

- 12 2) Output B .

13 NOTE—If both parties follow the algorithms specified, the value B computed here is $\hat{e}(Q_0, P_1)^t$, the same

14 as that obtained in P-IBE-E, above.

15 7.5 Pairing-based HIBS primitives

16 These primitives define operations with cryptographic strength defined by reduction to the Bilinear Diffie-
17 Hellman (BDH) problem, defined as:

18 Given group elements g_1, g_1^x, g_1^y, g_1^s , compute $\hat{e}(g_1, g_1)^{xys}$

19 For more information on this problem, the proof that reduces the problem of breaking these primitives to
20 BDH, and information on the associated primitives, see [BB2004].

21 For all these operations, the systems parameters are assumed to be defined as:

22 The server secrets are defined to be:

- 23 — s_0, s_1, \dots, s_n , random elements of Z_p
- 24 — The derived secret keys S_0, S_1, \dots, S_n , elements of G_2 , where $S_0 = O$

25 The public parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0, Q_1, \dots, Q_n)$ are defined to be:

- 26 — G_1, G_2, G_3, e , the system parameters as defined above in Section 6
- 27 — P_0 , a generator of G_2

- 1 — $Q_0 = s_0 P_0$
 2 — The auxiliary public parameters Q_1, Q_2, \dots, Q_n , elements of G_2

3
 4 The identity strings are defined to be:

- 5 — ID_0, ID_1, \dots, ID_n , strings

6 **7.5.1 Pairing-based HIBS: Generation (P-HIBS-G)**

7 Input:

- 8 — The parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0, Q_1, \dots, Q_n)$
 9 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-2}
 10 — Server secret s_{t-1} and a derived secret key S_{t-1}
 11 — Encoded identity P_t in G_2 , where P_t is typically derived from the identity string
 12 ID_0, ID_1, \dots, ID_t

13 Assumptions: parameters \mathcal{P} describe valid bilinear groups.

14 Output:

- 15 — Derived secret key S_t , element of G_2
 16 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} , elements of G_2

17 Operation: Use the following steps to compute S_t and Q_{t-1} .

- 18 1) Compute $S_t = S_{t-1} + s_{t-1} P_t$
 19 2) Compute $Q_{t-1} = s_{t-1} P_0$
 20 3) Output S_t and Q_1, Q_2, \dots, Q_{t-1}

21 **7.5.2 Pairing-based HIBS: Verification (P-HIBS-V)**

22 Input:

- 23 — Parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0, Q_1, \dots, Q_n)$
 24 — Auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} in G_2
 25 — Derived secret key S_{t-1}
 26 — Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is typically derived from the identity string
 27 ID_0, ID_1, \dots, ID_i
 28

- 1 Output:
- 2 — The value "valid" if secret key S_t and auxiliary public parameters Q_1, Q_2, \dots, Q_{t-1} are consistent
- 3 with P and P_i or "invalid" otherwise

4 Operation: Use the following steps.

- 5 1) Compute $T = \hat{e}(Q_0, S_t)$
- 6 2) Compute $V = \prod_{i=1}^t \hat{e}(Q_{i-1}, P_i)$
- 7 3) If $T = V$ then output the value "valid," otherwise output the value "invalid."

8 7.5.3 Pairing-based HIBS: Signature Generation (P-HIBS-SG)

9 Input:

- 10 — The parameters $\mathcal{P}=(G_1, G_2, G_3, e, P_0, Q_0, Q_1, \dots, Q_n)$
- 11 — Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is derived from the identity string
- 12 ID_0, ID_1, \dots, ID_i
- 13 — A message hash value P_M , an element of G_2
- 14 — Secret key S_t and Q_1, Q_2, \dots, Q_t

15 Assumptions: The parameters \mathcal{P} describe a valid bilinear group and public parameters; S_t and

16 Q_1, Q_2, \dots, Q_{t-1} are elements of G_2 .

17 Output:

- 18 — Signature Sig
- 19 — Q_1, Q_2, \dots, Q_t , elements in G_2

20 Operation: Use the following steps.

- 21 1) Compute $U_0 = rP_0$
- 22 2) Compute $Sig = S_t + s_t P_M$
- 23 3) Output Sig , and Q_1, Q_2, \dots, Q_t .

24 7.5.4 Pairing-based HIBS: Signature Verification (P-HIBS-SV)

25 Input:

- 26 — The parameters

- 1 — Encoded identities P_i in G_2 for $1 \leq i \leq t$, where P_i is derived from the identity string
 2 ID_0, ID_1, \dots, ID_t
 3 — A signature Sig and auxiliary input Q_1, Q_2, \dots, Q_t
 4 — A message hash value P_M , an element of G_2

5 Output:

- 6 — The value "valid" if Signature is valid, otherwise "invalid"

7 Operation: Use the following steps.

- 8 1) Compute $B = \hat{e}(Q_0, P_1) \hat{e}(Q_t, P_M) \prod_{i=2}^t \hat{e}(Q_{i-1}, P_i)$
 9 2) If $\hat{e}(P_0, Sig) = B$, output "valid", otherwise output "invalid".

10 7.6 Pairing-based Type1 Proxy Re-encryption primitives

11 These primitives define operations with cryptographic strength defined by reduction to the Decision
 12 Bilinear Diffie-Hellman (DBDH) problem, defined as:

13 Given group elements g, g^x, g^y, g^z, W decide if $W = e(g, g)^{abc}$

14 For more information on this problem, the proof that reduces the problem of breaking these primitives to
 15 DBDH, and information on the associated primitives, see [BB2004].

16 For all these operations, the systems parameters are assumed to be defined as:

17 The server secret that is defined to be:

- 18 — A random element s of Z_p .

19 The public parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$ are defined to be:

- 20 — G_1 , a group of prime order p
 21 — G_2 , a group of prime order p
 22 — $e : G_1 \times G_1 \rightarrow G_2$, a pairing
 23 — Q_1 , a generator of G_1
 24 — Q_2, Q_3 , random elements of G_1
 25 — $Q_4 = sQ_1$
 26 — $V = e(Q_2, Q_4)$

1 7.6.1 Pairing-based BB₂: Generation (P-BB2-G)

2 Input:

- 3 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$
- 4 — The server secret s
- 5 — A randomizer u , an integer of Z_p
- 6 — An encoded identity m of Z_p , typically derived from an identity string

7 Assumptions: The parameters \mathcal{P} describe a valid bilinear group and the public parameters; s is the server
8 secret corresponding to \mathcal{P} .

9 Output:

- 10 — The secret key elements $K_{0,m}$ and $K_{1,m}$ corresponding to m , two elements of G_1

11 Operation: Use the following steps.

- 12 1) Compute $K_{0,m} = sQ_2 + u(mQ_4 + Q_3)$.
- 13 2) Compute $K_{1,m} = uQ_1$.
- 14 3) Output $K_{0,m}$ and $K_{1,m}$.

15 7.6.2 Pairing-based BB₂: Verification (P-BB2-V)

16 Input:

- 17 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$.
- 18 — A public key element m of Z_p .
- 19 — The corresponding secret key elements $K_{0,m}$ and $K_{1,m}$, two elements of G_1 .

20 Assumptions: the parameters \mathcal{P} describe valid bilinear groups and correspond to the server secret s ;
21 $K_{0,m}$ and $K_{1,m}$ are elements corresponding to m , according to P-BB2-G.

22 Output:

- 23 — The value "valid" if $K_{0,m}$ and $K_{1,m}$ are consistent with \mathcal{P} and m and "invalid" otherwise

24 Operation: Use the following steps.

- 25 1) Select a random element R of G_2 .
- 26 2) Compute ciphertext elements $E_{0,m}$ and $E_{1,m}$, and a blinding factor B by running P-BB2-E with
27 the public key element m .

- 1 3) Derive the blinding factor B' by running P-BB2-D with the input $E_{0,m}$, $E_{1,m}$ and B computed in
 2 step 2.
 3 4) If $B' = B$ then output the value "valid," otherwise output the value "invalid."

4 **7.6.3 Pairing-based BB₂: Encryption (P-BB2-E)**

5 Input:

- 6 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$
 7 — A public key element m of Z_p .
 8 — A message randomizer r , an element of Z_p .

9 Assumptions: the parameters \mathcal{P} describe valid bilinear groups.

10 Output:

- 11 — The ciphertext elements $E_{0,m}$ and $E_{1,m}$, both elements of G_1 , along with a blinding factor B , an
 12 element of G_2 .

13 Operation: Use the following steps.

- 14 1) Compute $E_{0,m} = rQ_1$.
 15 2) Compute $E_{1,m} = r(mQ_4 + Q_3)$.
 16 3) Compute $B = V^r$.
 17 4) Output $E_{0,m}$, $E_{1,m}$ and B .

18 NOTE—The point multiplications in steps 1 and 2 and the field exponentiation in step 3 can be performed
 19 very efficiently using classic pre-computation algorithms. Pre-computation is viable and inexpensive
 20 because the base elements Q_1, Q_3, Q_4 and V , are all part of the public parameters and do not depend on the
 21 recipient identity.

22 **7.6.4 Pairing-based BB₂: Decryption (P-BB2-D)**

23 Input:

- 24 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$
 25 — A public key element m of Z_p .
 26 — The secret key elements $K_{0,m}$ and $K_{1,m}$ corresponding to m , two elements of G_1
 27 — The ciphertext elements $E_{0,m}$ and $E_{1,m}$, two elements of G_1

28 Assumptions: the parameters \mathcal{P} describe a valid bilinear group and public parameters associated with the
 29 server secret s ; $K_{0,m}$ and $K_{1,m}$ are elements corresponding to m , according to P-BB2-G.

1 Output:

2 — A blinding factor B , an element of G_2

3 Operation: Use the following steps.

4 1) Compute $B = e(E_{0,m}, K_{0,m}) / e(E_{1,m}, K_{1,m})$.

5 2) Output B .

6 NOTE—If both parties follow the algorithms specified, the value B computed here is V^r , the same as that
7 obtained in P-BB2-E, above.

8 7.7 Pairing-based EV primitives

9 7.7.1 Pairing-based EV: Generation (P-EV-G)

10 Input:

11 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$

12 Assumptions: the parameters \mathcal{P} describe valid bilinear groups.

13 Output:

14 — The private key elements k_0, k_1 and k_2 , three elements of Z_p .

15 — The corresponding public key elements L_0, L_1 and L_2 , three elements of G_1 .

16 Operation: Use the following steps.

17 1) Select three random elements k_0, k_1 and k_2 of Z_p .

18 2) Compute $L_0 = k_0 Q_1$, $L_1 = k_1 Q_3$ and $L_2 = k_2 Q_4$.

19 3) Output k_0, k_1, k_2 as the private key elements, and L_0, L_1, L_2 as the corresponding public key
20 elements.

21 7.7.2 Pairing-based EV: Encryption (P-EV-E)

22 Input:

23 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$

24 — The public key elements L_0, L_1 and L_2 , three elements of G_1 .

25 — A message randomizer r , an element of Z_p .

26 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; L_0, L_1 and L_2 are generated according to \mathcal{P} -
27 EV-G.

28 Output:

1 — The ciphertext elements E_0, E_1 and E_2 , three elements of G_1 , along with a blinding factor B , an
 2 element of G_2 .

3 Operation: Use the following steps.

4 1) Compute $E_0 = rL_0$.

5 2) Compute $E_1 = rL_1$.

6 3) Compute $E_2 = rL_2$.

7 4) Compute $B = V^r$.

8 5) Output E_0, E_1, E_2 and B .

9 7.7.3 Pairing-based EV: Decryption (P-EV-D)

10 Input:

11 — The parameters $\mathcal{P}=(G_1, G_2, e, P_0, Q_1, Q_2, Q_3, Q_4, V)$

12 — A secret key element k_2 of Z_p

13 — A ciphertext element E_2 of G_1

14 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; k_2 is an element generated according to P-
 15 EV-G; E_2 is an element generated according to P-EV-E.

16 Output:

17 — A blinding factor B , an element of G_2

18 Operation: Use the following steps.

19 1) Compute $B = e(E_2, Q_2)^{1/k_2}$.

20 2) Output B .

21 NOTE—If both parties follow the algorithms specified, the value B computed here is V^r , the same as that
 22 obtained in P-EV-E, above.

23 7.8 Pairing-based BR₁ primitives

24 7.8.1 Pairing-based BR₁: Generation (P-BR1-G)

25 Input:

26 — The parameters \mathcal{P}

27 — A public key element m of Z_p

- 1 — The secret key elements k_0, k_1 and k_2 , three elements of Z_p
- 2 — The secret key element $K_{1,m}$ of G_1 , corresponding to m .
- 3 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; k_0, k_1 and k_2 are elements generated
- 4 according to P-EV-G; $K_{1,m}$ is an element corresponding to m , generated according to P-BB2-G.

5 Output:

- 6 — The re-encryption key elements u_0, u_1 and U .

7 Operation: Use the following steps.

- 8 1) Set $u_0 = k_0$ and $u_1 = k_1$, two elements of Z_p .
- 9 2) Compute $U = -k_2 K_{1,m}$ of G_1 .
- 10 3) Output u_0, u_1 and U as the re-encryption key elements corresponding to m .

11

12 7.8.2 Pairing-based BR1: Re-encryption (P-BR1-RE)

13 Input:

- 14 — The parameters \mathcal{P}
- 15 — A public key element m of Z_p
- 16 — The re-encryption key elements u_0, u_1 of Z_p , and U of G_1
- 17 — The ciphertext elements E_0, E_1 and E_2 , three elements of G_1

18 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; u_0, u_1 and U are elements generated

19 according to P-BR1-G; E_0, E_1 and E_2 are elements generated according to P-EV-E.

20 Output:

- 21 — The ciphertext elements $E_{0,m}$ and $E_{1,m}$, two elements of G_1 , along with a re-encryption factor R ,
- 22 an element of G_2 .

23 Operation: Use the following steps.

- 24 1) Compute $E_{0,m} = -u_0 E_0$.
- 25 2) Compute $E_{1,m} = -u_1 E_1$.
- 26 3) Compute $R = e(U, mE_2)$.
- 27 4) Output $E_{0,m}, E_{1,m}$ and R .

1 7.8.3 Pairing-based BR1: Verification (P-BR1-V)

2 Input:

- 3 — The parameters \mathcal{P}
- 4 — The public key elements L_0, L_1 and L_2 , three elements of G_1 .
- 5 — The ciphertext elements E_0, E_1 and E_2 , three elements of G_1

6 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; L_0, L_1 and L_2 are elements generated
7 according to P-EV-G; E_0, E_1 and E_2 are elements generated according to P-EV-E.

8 Output:

- 9 — The value “valid” if E_0, E_1 and E_2 are consistent with P, L_0, L_1 and L_2 , “invalid” otherwise.

10 Operation: Use the following steps.

- 11 1) Compute $T_0 = e(E_0, L_2)$.
- 12 2) Compute $T_1 = e(E_2, L_0)$.
- 13 3) Compute $T_2 = e(E_2, L_1)$.
- 14 4) Compute $T_3 = e(E_1, L_2)$.
- 15 5) Output “valid” if $T_0 = T_1$ and $T_2 = T_3$, “invalid” otherwise.

16 7.9 Pairing-based Wang key agreement primitives

17 7.9.1 Pairing-based Wang key agreement: derive public key (P-WKA-D1)

18 This primitive derives an entity’s public key from its public identity string.

19 Input:

- 20 — Elliptic curve domain parameters q, a, b, r and G
- 21 — A hash function H
- 22 — An octet string ID representing a user’s identity
- 23 — The server secret α

24 Assumptions: The elliptic curve domain parameters q, a, b, r and G are valid

25 Output:

- 26 — The derived public key W , which is a generator of the elliptic curve group G ; or “error” (note that
27 this should generally be defined by computing $H(ID)$ first, and then map this element to a
28 generator of the group G).

1 **7.9.2 Pairing-based Wang key agreement: derive private key (P-WKA-D2)**

2 This primitive derives an entity's private key from a public key and a server secret

3 Input:

- 4 — Elliptic curve domain parameters q, a, b, r and G
- 5 — The server secret α
- 6 — The public key W , which is derived using the P-WKA-D1 primitive

7 Assumptions: elliptic curve domain parameters q, a, b, r and G are valid, the public key W is valid.

8 Output:

- 9 — The derived private key U , which is an element of G , or "error"

10 NOTE— $U = \alpha W$

11 As detailed in Section ???, an implementation of a primitive may make certain assumptions about its
 12 inputs, as listed with the specification of each primitive. For example, if elliptic curve domain parameters
 13 and a public key are inputs to a primitive, the implementation may generally assume that the domain
 14 parameters are valid. The behavior of an implementation is unconstrained in the case that W is not an
 15 appropriate public key, and in such a case the implementation may or may not output an error condition. It
 16 is up to the properly implemented scheme to ensure that only appropriate inputs are passed to a primitive,
 17 or to accept the risks of passing inappropriate inputs.

18 **7.9.3 Pairing-based Wang key agreement: derive secret value (P-WKA-D3)**

19 This primitive derives a shared secret value from one party's two key pairs and another party's two public
 20 keys. If two parties correctly execute this primitive, they will produce the same output. This primitive can
 21 be invoked by a scheme to derive a shared secret key; specifically, it may be used with the scheme W-KA.
 22 It assumes that the input keys are valid.

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

25 Input:

- 26 — The elliptic curve domain parameters q, a, b, r and G ; associated keys $U, (u, V), W', V'$ (the
 27 domain parameters shall be the same for these keys)
- 28 — A pairing function e
- 29 — The party's own identity based private key U which is an elliptic curve point
- 30 — The party's own second key pair (u, V) , where $V = (x, y)$
- 31 — The other party's identity based public key W'
- 32 — The other party's second public key $V' = (x', y')$
- 33 — A hash function H that maps an octet string to an octet string of length h .

1 Assumptions: private key W , key pair (u, V) , public keys W', V' , and elliptic curve domain parameters
 2 q, a, b, r and G are valid; all the keys are associated with the same domain parameters

3 Output:

4 — The derived shared secret value, which is a nonzero field element z in $GF(q)$ or “error”

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

7 1) Convert x into an octet string o using FE2OSP.

8 2) Convert x' into an octet string o' using FE2OSP.

9 3) Compute an octet string $s = H(o, o')$

10 4) Convert s into an integer i using OS2IP

11 5) Compute an octet string $s' = H(o', o)$

12 6) Convert s' into an integer i' using OS2IP

13 7) Compute an elliptic curve point $P = (u + i)U$

14 8) Compute an elliptic curve point $Q = i'W' + V'$

15 9) Let $z = e(P, Q)$

16 10) If $z = 1$, then terminate

17 11) Output z as the shared secret value.

18 A conformance region should include:

19 — At least one valid set of elliptic curve domain parameters q, a, b, r and G

20 — At least one valid private key U for each set of domain parameters

21 — All valid key pairs (u, V) associated with the same set of domain parameters as s

22 — All valid public keys w' and v' associated with the same set of domain parameters as s

23 NOTE—This primitive does not address small subgroup attacks, which may occur when the public keys W' and
 24 V' are not valid and when Weil pairing is used. To prevent them, a key agreement scheme should validate the public
 25 keys W' and V' before executing this primitive.

26 7.10 Pairing-based SCK key agreement primitives

27 7.10.1 Pairing-based SCK key agreement: derive secret value (P-SCK-D1)

28 P-SCK-D1 is the identity-based secret value derivation primitive, SCK version. It is based on the work of
 29 [Smart2002, ChenKudla2002]. This primitive derives a shared secret value from a domain public key, one
 30 party's two key pairs and another party's two public keys. If two parties correctly execute this primitive,
 31 they will produce the same output. It assumes that the input keys are valid.

1 Input:

2 — Elliptic curve domain parameters q, a, b, r and G associated keys R, d, x, Q, E (the domain
3 parameters shall be the same for these keys)

4 — A pairing function e

5 — A domain public key R which is an elliptic curve point

6 — The party's own identity based private key d which is an elliptic curve point

7 — The party's own second private key x , which is an integer

8 — The other party's identity based public key Q which is an elliptic curve point

9 — The other party's second public key E which is an elliptic curve

10 Assumptions: The domain public key R , private key d , private key x , public keys Q and E , and elliptic
11 curve domain parameters q, a, b, r and G are valid; all the keys are associated with the domain parameters.

12 Output:

13 — The derived shared secret value, which is a nonzero field element z in $GF(q)$; or "error".

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

16 1) Compute an elliptic curve point $A = xQ$,

17 2) Compute a pairing $t_1 = e(A, R)$

18 3) Convert t_1 into an octet string o_1 using FE2OSP,

19 4) Compute a pairing $t_2 = e(d, E)$

20 5) Convert t_2 into an octet string o_2 using FE2OSP,

21 6) Compute an elliptic curve point $B = xE$

22 7) Convert B into an octet string o using EP2OSP [this primitive should have been specified
23 somewhere]

24 8) Let $z = o || o_1 || o_2$ [how to address this operation should have been specified somewhere?]

25 9) If $z = I$ [what?] then terminate,

26 10) Output z as the shared secret value.

27 A conformance region should include:

28 — At least one valid set of elliptic curve domain parameters q, a, b, r and g

29 — At least one valid private key d for each set of domain parameters

30 — A valid private key x associated with the same set of domain parameters as s

31 — All valid public keys Q and E associated with the same set of domain parameters as s

32 NOTE—This primitive does not address small subgroup attacks, which may occur when the public keys Q and E are
33 not valid. To prevent them, a key agreement scheme should validate the public keys Q and E before executing this
34 primitive.

1 7.11 Pairing-based Type-2 Proxy Re-encryption primitives

2 For all these operations, the system parameters are assumed to be defined as the exactly same as that in
3 section 7.9.

4 7.11.1 Pairing-based BR2: Generation (P-BR2-G)

5 Input:

- 6 — The parameters \mathcal{P}
- 7 — The server secret s
- 8 — The secret key element $K_{1,m}$ of G_1

9 Assumptions: the parameters \mathcal{P} describe a valid bilinear group and are the public parameters associated
10 with the server secret s ; $K_{1,m}$ is an element corresponding to the public key element m of Z_p , according to
11 P-BB2-G.

12 Output:

- 13 — The re-encryption key element U of G_1 .

14 Operation: Use the following steps.

- 15 1) Compute $U = sK_{1,m}$ of G_1 .
- 16 2) Output U as the re-encryption key element corresponding to m .

17 7.11.2 Pairing-based BR2: Re-encryption (P-BR2-RE)

18 Input:

- 19 — The parameters \mathcal{P}
- 20 — The public key elements m_0 and m_1 , two elements of Z_p
- 21 — The re-encryption key element U of G_1
- 22 — The ciphertext elements E_{0,m_0}, E_{1,m_0} corresponding to m_0 , two elements of G_1

23 Assumptions: the parameters \mathcal{P} describe valid bilinear groups and are the public parameters associated with
24 the server secret s ; U is an element corresponding to m_1 , according to P-BR2-G; E_{0,m_0} and E_{1,m_0} are
25 elements according to P-BB2-E.

26 Output:

- 27 — The ciphertext elements E_{0,m_1} and E_{1,m_1} , two elements of G_1 , along with a re-encryption factor R ,
28 an element of G_2 .

29 Operation: Use the following steps.

- 1 1) Set $E_{0,m_1} = E_{0,m_0}$ and $E_{1,m_1} = E_{1,m_0}$.
- 2 2) Compute $m = m_1 - m_0$ in Z_p .
- 3 3) Compute $R = e(U, mE_{0,m_0})$.
- 4 4) Output E_{0,m_1}, E_{1,m_1} and R .

5 7.11.3 Pairing-based BR2: Verification (P-BR2-V)

6 Input:

- 7 — The parameters \mathcal{P}
- 8 — A public key element m in Z_p
- 9 — The ciphertext elements $E_{0,m}, E_{1,m}$ corresponding to m , two elements of G_1

10 Assumptions: the parameters \mathcal{P} describe valid bilinear groups and are the public parameters associated with
 11 the server secret s ; $E_{0,m}$ and $E_{1,m}$ are elements according to P-BB2-E.

12 Output:

- 13 — The value “valid” if $E_{0,m}, E_{1,m}$ are consistent with P and m , “invalid” otherwise.

14 Operation: Use the following steps.

- 15 — Compute $T_0 = e(E_{0,m}, mQ_4)$.
- 16 — Compute $T_1 = e(E_{0,m}, Q_3)$.
- 17 — Compute $T_2 = e(E_{1,m}, Q_1)$.
- 18 — Output “valid” if $T_0T_1 = T_2$, “invalid” otherwise.

19 8. Pairing-based Identity-based Encryption Schemes

20 This section describes Identity-based Encryption (IBE) methods and Identity-based Key Encapsulation
 21 Mechanisms (ID-KEM) for each of the family of primitives described in Section 6. The setting is the same
 22 as the primitives, in that the schemes assume that there are three parties involved in the operations: a key
 23 server, a sender and a recipient. Each IBE scheme is described in terms of four operations: setup, extract,
 24 encrypt and decrypt. Each ID-KEM scheme is described with the same setup and extract operations, but
 25 with encapsulate and decapsulate operations instead of encryption and decryption.

26 The IBE and ID-KEM operations are similar, but have different performance characteristics suited to
 27 different applications. IBE has the ability to directly encrypt a message, while ID-KEM provides the
 28 ability to encrypt to multiple recipients in a more efficient way.

1 **8.1 Operations**

2 **8.1.1 Shared operations**

3 The following operations are common to IBE and ID-KEM schemes.

4 Setup: This operation is performed at the key server. This operation generates the secret and public
 5 parameters for the key server and establishes the parameters for all subsequent operations using that key
 6 server. This standard does not describe the necessary security design elements needed to protect the key
 7 server's secret keying material, or the mechanisms needed to transmit the public parameter information to
 8 the sender and recipient. These are both crucial elements, but are dependent on the specific security
 9 requirements of the application.

10 Extract: This operation is performed at the key server, typically after an authenticated request from the
 11 recipient. This operation takes an identity string ID and uses the key server secret to compute the private
 12 key corresponding to the identity string. This standard does not describe the authentication mechanism
 13 used to insure that private keys are only given to the proper recipients. The specific methods and protocols
 14 used for authentication are critical to system security, but are dependent on the requirements of the
 15 application.

16 **8.1.2 IBE Operations**

17 The following operations are unique to IBE schemes.

18 Encrypt: This operation is performed at the sender. Given an identity string, a set of public parameters
 19 from a key server, and a message m , this operation encrypts the message. Note that, in practice, the
 20 message m will be a session key used to encrypt a larger message, although this is not mandatory.

21 Decrypt: This operation is performed at the recipient. Given a message encrypted with a set of server
 22 parameters and identity string ID , this operation uses the corresponding private key K_{ID} , generated during
 23 an Extract operation, to decrypt the message.

24 **8.1.3 ID-KEM Operations**

25 The following operations are unique to ID-KEM schemes.

26 Encapsulate: This operation is performed at the sender. Given an identity string ID and a set of public
 27 parameters from a key server, this operation outputs a pair (k, c) where k is an encryption key used to
 28 encrypt the message with a symmetric algorithm, and c is the encapsulation of key k . The encapsulation c
 29 and the message encrypted with k are typically delivered to the recipient.

30 Decapsulate: This operation is performed at the recipient. Given an encapsulation c encrypted with an
 31 identity string ID , and the corresponding private key K_{ID} , generated during an Extract operation, returns
 32 the decryption key k .

1 8.2 The pairing-based SK Scheme

2 8.2.1 SK-KEM

3 SK-KEM uses the SK family of primitives to construct an ID-KEM mechanism that has a security
4 reduction to the q-BSK problem. This KEM is based on the work of Sakai and Kasahara [SAKAI03], and
5 is described in full with security proofs in [CHEN05]. The scheme takes a security parameter t , and a key
6 size parameter n .

7 It requires the definition of four hash functions:

8 — $H_1 : \{0,1\}^* \rightarrow Z_p$ where $H_1(s) = IHF1(BS2OSP(s), p, t)$

9 — $H_2 : G_3 \rightarrow \{0,1\}^n$ where $H_2(x) = SHF1(FE2OSP(x), n, t)$

10 — $H_3 : \{0,1\}^* \rightarrow Z_q$ where $H_3(s) = IHF1(BS2OSP(s), q, t)$

11 — $H_4 : \{0,1\}^* \rightarrow \{0,1\}^n$ where $H_4(s) = SHF1(BS2OSP(s), n, t)$

12 The scheme also requires a random bit generation algorithm:

13 — R_1 : a source of random values in the space $\{0,1\}^n$.

14 8.2.1.1 SK KEM: Setup (SK-KEM-S)

15 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
16 steps. This operation creates a server secret which must be secured, as all private keys calculated within
17 the system depend on it. It is recommended that applications establish a methodology for changing the
18 server secret and public parameters on a regular basis, and have a methodology for handling the disclosure
19 of a server secret.

20 The steps to setup the key server and system parameters are:

- 21 1) Establish the set of base groups G_1, G_2, G_3 , and a pairing $e : G_1 \times G_2 \rightarrow G_3$. Annex X contains
22 recommendations for the generation of these objects.
- 23 2) Select a random generator Q_1 in G_1 using A.X.X. Calculate the corresponding generator Q_2 in G_2 .
- 24 3) Generate a random server secret s in Z_p^* . Calculate the corresponding R as sQ_1 .
- 25 4) Pre-calculate the pairing value $O = e(Q_1, Q_2)$.
- 26 5) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$ available. Secure the server
27 secret s in a way appropriate to the application.

28 8.2.1.2 SK KEM: Extract (SK-KEM-EX)

29 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding private
30 key K_{ID} in G_2 . It is recommended that applications establish a methodology for authenticating access to
31 private keys by using the ID string as an identity in a trusted authentication system. The details of
32 authenticating the key request are beyond the scope of this document, but are critical for the security of an
33 implemented application.

1 The steps to compute the private key K_{ID} corresponding to an identity string ID are:

- 2 1) Compute the identity element $M = H_1(ID)$.
- 3 2) Use the P-SK-G primitive to compute K_{ID} using M and the server secret s and public system
- 4 parameters.

5 **8.2.1.3 SK KEM: Encapsulate (SK-KEM-EN)**

6 The encapsulate operation takes an arbitrary identity string ID in $\{0,1\}^*$ and the public parameters for a key
7 server, and outputs the pair (k, c) , where k is a key to be used to encrypt a message, and c is the
8 encapsulation of k to be transmitted to the receiver.

9 The steps to compute the encapsulation values are:

- 10 1) Using R_1 , generate a random n -bit message m .
- 11 2) Compute $r = H_3(m)$.
- 12 3) Compute P-SK-E, using r as the message randomizer.
- 13 4) Output $c = (E, B \oplus m)$.
- 14 5) Output $k = H_4(m)$.

15 **8.2.1.4 SK KEM: Decapsulate (SK-KEM-DE)**

16 The decapsulate operation takes an encapsulated value c computed for identity ID , and the private key K_{ID}
17 that corresponds to ID , and computes the key value k that can be used to decrypt the message that was
18 encrypted by the sender.

19 The steps to compute the decapsulated key are:

- 20 1) Parse the encapsulated value c as the pair (U, V) .
- 21 2) Compute P-SK-D on U , returning the value B .
- 22 3) Compute $m = H_2(B) \oplus V$.
- 23 4) Compute $r = H_3(m)$.
- 24 5) Compute $Q = R + (H_1(ID) \cdot Q_1)$.
- 25 6) Verify that $U = rQ$. If not, output "error" and stop.
- 26 7) Return $k = H_4(m)$.

1 8.3 The BB1 Scheme

2 8.3.1 BB1-KEM

3 BB1-KEM uses the P-BB1 family of primitives to construct a KEM mechanism that has a security
4 reduction to the BDH problem. This KEM is based on the work of Boneh and Boyen [BB04], and is
5 explicitly described in [BOYEN06]. The scheme takes a security parameter t , and a key size parameter n .

6 It requires the definition of two hash functions:

7 — $H_1 : \{0,1\}^* \rightarrow Z_p$, where $H_1(s) = IHF1(BS2OSP(s), p, t)$

8 — $H_2 : G_3 \rightarrow \{0,1\}^n$, where $H_2(x) = SHF1(FE2OSP(x), n, t)$

9 The scheme also requires a random bit generation algorithm:

10 — R_1 : a source of random values in the space Z_p

11 8.3.1.1 BB₁ KEM: Setup (BB1-KEM-S)

12 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
13 steps. This operation creates a server secret which must be secured, as all private keys calculated within
14 the system depend on it. It is recommended that applications establish a methodology for changing the
15 server secret and public parameters on a regular basis, and have a methodology for handling the disclosure
16 of a server secret.

17 The steps to setup the key server and system parameters are:

- 18 1) Establish the set of base groups G_1, G_2, G_3 and a pairing $e : G_1 \times G_2 \rightarrow G_3$. Annex X contains
19 recommendations for the generation of these objects.
- 20 2) Select a random generator Q_1 in G_1 using A.X.X.
- 21 3) Select a random generator Q_2 in G_2 which may, but does need not be, related to Q_1 by an explicit
22 mapping ϕ .
- 23 4) Generate random server secrets s_1, s_2 and s_3 in Z_p .
- 24 5) Calculate the corresponding R as $s_1 Q_1$ and T as $s_3 Q_1$.
- 25 6) Calculate the pairing value V as $e(s_1 Q_1, s_2 Q_2)$
- 26 7) Make the public parameter set $\mathcal{P} = (Q_1, Q_2, R, T, V, G_1, G_2, e)$ available. Secure the server
27 secrets s_1, s_2 and s_3 in a way appropriate to the application.

28 8.3.1.2 BB₁ KEM: Extract (BB1-KEM-EX)

29 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding private
30 key elements $K_{0,ID}$ and $K_{1,ID}$ in G_2 . It is recommended that applications establish a methodology for
31 authenticating access to private keys by using the ID string as an identity in a trusted authentication system.
32 The details of authenticating the key request are beyond the scope of this document, but are critical for the
33 security of an implemented application.

1 The steps to compute the private key $K_{0,ID}$ and $K_{1,ID}$ corresponding to an identity string ID are:

- 2 1) Compute the identity element $M = H_1(ID)$.
- 3 2) Use the P-BB1-G primitive to compute $K_{0,ID}$ and $K_{1,ID}$ using M , the server secrets s_1 , s_2 and s_3 ,
- 4 the public system parameters.

5 **8.3.1.3 BB₁ KEM: Encapsulate (BB1-KEM-EN)**

6 The encapsulate operation takes an arbitrary identity string ID in $\{0,1\}^*$ and the public parameters for a key
7 server, and outputs the pair (k, c) where k is a key to be used to encrypt a message, and c is the
8 encapsulation of k to be transmitted to the receiver.

9 The steps to compute the encapsulation values are:

- 10 1) Using R_1 , generate a random integer r .
- 11 2) Compute P-BB1-E, using r as the message randomizer.
- 12 3) Output c as the pair (E_0, E_1) .
- 13 4) Output $k = H_2(B)$.

14 **8.3.1.4 BB₁ KEM: Decapsulate (BB1-KEM-DE)**

15 The decapsulate operation takes an encapsulated value c computed for identity ID , and the private key
16 $K_{0,ID}$ and $K_{1,ID}$ that corresponds to ID , and computes the key value k that can be used to decrypt the
17 message that was encrypted by the sender.

18 The steps to compute the decapsulated key are:

- 19 1) Parse the encapsulated value c as the pair (E_0, E_1) .
- 20 2) Compute PBB1D on (E_0, E_1) to obtain the value B .
- 21 3) Return $k = H_2(B)$.

22 **8.3.2 BB₁ IBE**

23 BB₁ IBE uses the P-BB1 family of primitives to construct an IBE mechanism that has a security reduction
24 to the BDH problem. This cryptosystem is based on the work of Boneh and Boyen [BB04], and is
25 explicitly described in [BOYEN06]. The scheme takes a security parameter t , and a key size parameter n .

26 It requires the definition of three hash functions:

- 27 — $H_1 : \{0,1\}^* \rightarrow Z_p$, where $H_1(s) = IHF1(BS2OSP(s), p, t)$
- 28 — $H_2 : G_3 \rightarrow \{0,1\}^n$, where $H_2(x) = SHF1(FE2OSP(x), n, t)$
- 29 — $H_3 : G_3 \times \{0,1\}^n \times G_1 \times G_1 \rightarrow Z_p$,
- 30 where $H_3(x, s, P_1, P_2) = IHF1(FE2OSP(x) || BS2OSP(s) || EC2OSP(P_1) || EC2OSP(P_2), p, t)$

1 The scheme also requires a random bit generation algorithm:

2 — R_1 : a source of random values in the set Z_p

3 **8.3.3 BB₁ IBE: Setup (BB1-IBE-S)**

4 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
5 steps. This operation creates a server secret which must be secured, as all private keys calculated within
6 the system depend on it. It is recommended that applications establish a methodology for changing the
7 server secret and public parameters on a regular basis, and have a methodology for handling the disclosure
8 of a server secret.

9 The steps to setup the key server and system parameters are:

- 10 1) Establish the set of base groups G_1, G_2, G_3 and a pairing $e : G_1 \times G_2 \rightarrow G_3$. Annex X contains
11 recommendations for the generation of these objects.
- 12 2) Select a random generator Q_1 in G_1 using using A.X.X.
- 13 3) Select also a random generator Q_2 in G_2 , which may but need not be related to Q_1 by an explicit
14 mapping ϕ .
- 15 4) Generate random server secrets s_1, s_2 and s_3 , in Z_p .
- 16 5) Calculate the corresponding R as $s_1 Q_1$ and T as $s_3 Q_1$.
- 17 6) Calculate the pairing value V as $e(s_1 Q_1, s_2 Q_2)$.
- 18 7) Make the public parameter set $\mathcal{P} = (Q_1, Q_2, R, T, V, G_1, G_2, e)$ available. Secure the server
19 secrets s_1, s_2 and s_3 in a way appropriate to the application.

20 **8.3.4 BB₁ IBE: Extract (BB1-IBE-EX)**

21 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding private
22 key elements $K_{0,ID}$ and $K_{1,ID}$ in G_2 . It is recommended that applications establish a methodology for
23 authenticating access to private keys by using the ID string as an identity in a trusted authentication system.
24 The details of authenticating the key request are beyond the scope of this document, but are critical for the
25 security of an implemented application.

26 The steps to compute the private key $K_{0,ID}$ and $K_{1,ID}$ corresponding to an identity string ID are:

- 27 1) Compute the identity element $M = H_1(ID)$.
- 28 2) Use the P-BB1-G primitive to compute $K_{0,ID}$ and $K_{1,ID}$ using M , the server secrets $K_{0,ID}$ and
29 $K_{1,ID}$ using M , the server secrets s_1, s_2, s_3 , and the public system parameters.

30 **8.3.4.1 BB₁ IBE: Encrypt (BB1-IBE-EN)**

31 The encryption operation takes an arbitrary identity string ID in $\{0,1\}^*$ a message M in $\{0,1\}^*$, and the
32 public parameters for a key server, and outputs a ciphertext c to be transmitted to the receiver.

1 The steps to compute the ciphertext are:

- 2 1) Using R_1 , generate a random integer r in Z_p .
- 3 2) Compute P-BB1-E, using r as the message randomizer.
- 4 3) Compute $Y = H_2(B) \oplus M$.
- 5 4) Compute $t = r + H_3(B, Y, E_0, E_1)$ in Z_p .
- 6 5) Output the quadruple $c = (Y, E_0, E_1, t)$ as the ciphertext.

7 8.3.4.2 BB₁ IBE: Decrypt (BB1-IBE-DE)

8 The decryption operation takes a ciphertext c computed for identity ID , and the private key $K_{0,ID}$ and
 9 $K_{1,ID}$ that corresponds to ID , and computes the message M that was encrypted by the sender, or an
 10 “error” signal.

11 The steps to compute the decapsulated key are:

- 12 1) Parse c as the quadruple $c = (Y, E_0, E_1, t)$.
- 13 2) Compute PBB1D on (E_0, E_1) , to obtain the value B .
- 14 3) Compute $r = t - H_3(B, Y, E_0, E_1)$ in Z_p .
- 15 4) Verify that $B = V^r$ and that $E_0 = rQ_1$. If not, output “error” and stop.
- 16 5) Return $M = Y \oplus H_2(B)$.

17 NOTE—the verifications in step 4 can be performed very efficiently using classic pre-computation methods for field
 18 exponentiation and curve multiplication. Pre-computation is viable because the base elements V and Q_1 are part of the
 19 public parameters and thus constant and independent of the recipient identity.

20 8.4 The BF Scheme

21 8.4.1 BF IBE

22 BF IBE uses the P-BF family of primitives to construct an IBE scheme that has a security reduction to the
 23 Bilinear Diffie-Hellman (BDH) problem. This IBE scheme is based on the work of Boneh and Franklin
 24 [Boneh2001] and is described with full security proofs in that work. The scheme takes a security parameter
 25 t , and a key size parameter n . The following scheme uses the Fujisaki-Okamoto transform to create an
 26 IND-CCA secure IBE scheme.

27 BF-IBE requires the definition of four hash functions:

- 28 — $H_1 : \{0,1\}^* \rightarrow G_1$, where $H_1(s) = PHF1(BS2OSP(s), t, j, q, p)$
- 29 — $H_2 : G_3 \rightarrow \{0,1\}^n$, where $H_2(x) = SHF1(FE2OSP(x), n, t)$
- 30 — $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow Z_p^*$, where $H_3(s_1, s_2) = IHF1(BS2OSP(s_1) \parallel BS2OSP(s_2), p-1, t)$

1 — $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$, where $H_4(s) = SHF1(BS2OSP(s), n, t)$

2 The scheme also requires a random bit generation algorithm:

3 — R_1 : a source of random bits in $\{0,1\}^n$

4 **8.4.1.1 BF IBE: Setup (BF-IBE-S)**

5 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
6 steps. This operation creates a server secret which must be secured, as all private keys calculated within
7 the system depend on it. It is recommended that applications establish a methodology for changing the
8 server secret and public parameters on a regular basis, and have a methodology for handling the disclosure
9 of a server secret.

10 The steps to setup the key server and system parameters are:

- 11 1) Establish the set of base groups G_1, G_3 , and a pairing $e : G_1 \times G_1 \rightarrow G_3$. Annex X contains
12 recommendations for the generation of these objects. G_1 must be an element of $E(GF(q))[p]$,
13 where p and q are primes and E is
14 either $E/GF(q) : y^2 = x^3 + 1$ or $E/GF(q) : y^2 = x^3 + x$, $p \mid (q+1)$ and $p^2 \nmid (q+1)$.
- 15 2) Select a random generator Q in G_1 using A.X.X.
- 16 3) Generate a random server secret s in Z_p^* . Calculate the corresponding R as sQ .
- 17 4) Make the public parameter set $P = (R, Q, G_1, G_3, e)$ available. Secure the server secret s in a
18 way appropriate to the application.

19 **8.4.1.2 BF IBE: Extract (BF-IBE-EX)**

20 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding
21 private key K_{ID} in G_1 . It is recommended that applications establish a methodology for authenticating
22 access to private keys by using the ID string as an identity in a trusted authentication system. The details of
23 authenticating the key request are beyond the scope of this document, but are critical for the security of an
24 implemented application.

25 The steps to compute the private key K_{ID} corresponding to an identity string ID are:

- 26 1) Compute the identity element $M = H_1(ID)$.
- 27 2) Use the P-BF-G primitive to compute K_{ID} using M and the server secret s and public system
28 parameters.

29 **8.4.1.3 BF IBE: Encrypt (BF-IBE-EN)**

30 The encrypt operation takes an arbitrary identity string ID in $\{0,1\}^*$, the set of public parameters \mathcal{P} , and a
31 message m of length n , and calculates the ciphertext E of the message. The intention is that only the
32 possessor of the corresponding private key K_{ID} will be able to decrypt E to recover m . Note that, in
33 practice, m will typically be a symmetric encryption key used to encrypt a larger data block typically
34 transmitted or stored with E .

1 The steps to compute the ciphertext E are:

- 2 1) Using R_1 , compute an n -bit message randomizer o .
- 3 2) Compute $M = H_1(ID)$.
- 4 3) Compute $r = H_3(o, M)$.
- 5 4) Compute $C_1 = rQ$.
- 6 5) Compute the blinding value B using P-BF-EN with M , r and the public parameters \mathcal{P} .
- 7 6) Compute $C_2 = o \oplus H_2(B)$.
- 8 7) Compute $C_3 = m \oplus H_4(o)$.
- 9 8) Output the ciphertext $E = (C_1, C_2, C_3)$.

10 8.4.1.4 BF IBE: Decrypt (BF-IBE-DE)

11 The decrypt operation takes a ciphertext E and the private key K_{ID} from the FDH-IBE-EX operation and
12 either recovers the message m or outputs “error.”

13 The steps to decrypt E into m are the following:

- 14 1) Using P-BF-D with C_1 as the ciphertext, and K_{ID} as the key, compute the blinding value B .
- 15 2) Compute $o = C_2 \oplus H_2(B)$.
- 16 3) Compute $m = C_3 \oplus H_4(o)$.
- 17 4) Compute $r = H_3(o, M)$.
- 18 5) Test if $C_1 = rQ$. If not, output “error” and stop.
- 19 6) Output m as the decrypted message

20 9. Pairing-based Signature Schemes

21 9.1 DHI Signature

22 DHI Signature uses the DHI family of primitives to construct an identity-based signature that has a security
23 reduction to, an assumption which is related to (and actually weaker than) the q -BDHI assumption. This
24 signature is actually a non-interactive proof of knowledge of a private key generated according to the work
25 of Sakai and Kasahara [SAKAI03].

26 It requires the definition of two hash functions:

27 — $H_1 : \{0,1\}^* \rightarrow Z_p$

28 — $H_2 : \{0,1\}^* \rightarrow Z_q$

1 9.1.1 DHI Signature: Setup (DHI-SIG-S)

2 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
3 steps. This operation creates a server secret which must be secured, as all private keys calculated within
4 the system depend on it. The steps to setup the key server and system parameters are:

5 Input:

6 — Some stuff

7 Output:

8 — Some stuff

9 The steps to setup the key server and public parameters \mathcal{P} are:

- 10 1) Establish the set of base groups G_1, G_2, G_3 , and a pairing $e : G_1 \times G_2 \rightarrow G_3$.
- 11 2) Pick a random generator Q_2 in G_2 . Calculate the corresponding $Q_1 = \phi(Q_2)$ in G_1 .
- 12 3) Generate a random server secret s in Z_p^* . Calculate the corresponding R as sQ_2 .
- 13 4) Pre-calculate the pairing value $O = e(Q_1, Q_2)$.
- 14 5) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$ available. Secure the server secret
15 s in a way appropriate to the application.

16 9.1.2 DHI Signature: Extract (DHI-SIG-EX)

17 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding private
18 key K_{ID} in G_1 . To minimize the size of signatures and private keys, it is advisable to set up users' keys in
19 G_1 instead of G_2

20 Input:

21 — Some stuff

22 Output:

23 — Some stuff

24 The steps to compute the private key K_{ID} corresponding to an identity string ID are:

- 25 1) Compute the identity element $M = H_1(ID)$ in Z_p .
- 26 2) Use the P-DHI-G primitive to compute $K_{ID} = (M + s)^{-1} Q_1$ using the server secret s .

27 9.1.3 DHI Signature: Create Signature (DHI-SIG-SI)

28 Input:

29 — The parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$

30 — The public key element M

- 1 — The associated private key $K_{ID} = (M + s)^{-1}Q_1$
- 2 — A randomizer r , an integer
- 3 — A message m to be signed
- 4 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is the public parameter associated with s ;
- 5 M is an element of Z_p .

6 Output:

- 7 — A signature (h, S) where h is an element of Z_p and S is an element of G_1 .

8 Operation: Use the following steps.

- 9 1) Compute $u = e(Q_1, Q_2)^r$.
- 10 2) Compute $h = H_2(m, u)$.
- 11 3) Compute $S = (r + h)K_m$
- 12 4) Output h and S .

13 9.1.4 DHI Signature: Verify Signature (DHI-SIG-VE)

14 Input:

- 15 — The parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$
- 16 — The public key element M
- 17 — A signature (h, S)

18 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is a public key, an element of G_2 ; M is an

19 element of Z_p ; K_M , an element of G_1 , is a valid generated value for M according to P-DHI-GV; h is an

20 element of Z_p and S is an element of G_1 .

21 Output:

- 22 — The value "valid" if the purported signature (h, S) is accepted with respect to the public key
- 23 element M and "invalid" otherwise.

24 Operation: Use the following steps.

- 25 1) Compute $u = \frac{e(S, MQ_2 + R)}{e(Q_1, Q_2)^h}$.
- 26 2) Output "valid" if $h = H_2(m, u)$ and "invalid" otherwise.

27 If both parties follow the algorithms specified, a valid signature (h, S) is always accepted.

1 10. Pairing-based Signcryption Schemes

2 10.1 DHI Signcryption

3 DHI Signcryption uses the DHI family of primitives to construct a fast scheme to jointly perform both a
4 signature and encryption in settings that require private and authenticated communications. The scheme has
5 security proofs under the q -BDHI assumption. This construction efficiently combines the DHI signature
6 with the DHI-like encryption layer. The scheme is used to sign and encrypt messages of length n .

7 It requires the definition of three hash functions:

8 — $H_1 : \{0,1\}^* \rightarrow Z_p$

9 — $H_2 : \{0,1\}^* \rightarrow Z_q$

10 — $H_3 : G_3 \rightarrow \{0,1\}^n$

11 10.1.1 DHI Signcryption: Setup (DHI-SC-S)

12 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
13 steps.

14 The steps to setup the key server and system-wide parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$ are:

- 15 1) Establish the set of base groups G_1, G_2, G_3 , and a pairing $e : G_1 \times G_2 \rightarrow G_3$.
- 16 2) Pick a random generator Q_2 in G_2 . Calculate the corresponding $Q_1 = \phi(Q_2)$ in G_1 .
- 17 3) Generate a random server secret s in Z_p^* . Calculate the corresponding R as sQ_2 .
- 18 4) Pre-calculate the pairing value $O = e(Q_1, Q_2)$.
- 19 5) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$ available. Secure the server secret
20 s in a way appropriate to the application.

21 10.1.2 DHI Signcryption: Extract (DHI-SC-EX)

22 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$ and calculates the corresponding
23 private key K_{ID} in G_2 . This time users' keys must lie in G_2 . In order to minimize the size of ciphertexts, the
24 isomorphism $\phi : G_2 \rightarrow G$ will come into play in the protocol.

25 The steps to compute the private key K_{ID} corresponding to an identity string ID are:

- 26 1) Compute the identity element $M = H_1(ID)$ in Z_p .
- 27 2) Use the P-DHI-G primitive to compute $K_{ID} = (M + s)^{-1} Q_2$ using the server secret s .

1 **10.1.3 DHI Signcryption: Sign and Encrypt (DHI-SC-SE)**

2 Input:

3 — The parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$

4 — The sender's public identifier M_A

5 — The associated private key $K_A = (M_A + s)^{-1}Q_2$

6 — The receiver's identifier M_B

7 — A randomizer r , an integer

8 — A message m to be jointly signed and encrypted

9 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is the public parameter associated with s ;
10 M_A and M_B are both elements of Z_p .

11 Output:

12 — A ciphertext (c, S, T) , where c is a bitstring of length n and S and T are both elements of G_1 .

13 Operation: Conduct the following steps.

14 1) Compute $u = e(Q_1, Q_2)^r$.

15 2) Compute $h = H_2(m, u)$.

16 3) Compute $S = (r + h)K_A$.

17 4) Compute $T = r(M_B + \phi(R))$.

18 5) Compute $c = m \oplus H_3(u)$

19 6) Output the ciphertext (c, S, T)

20 **10.1.4 DHI Signcryption: Decrypt and Verify (DHI-SC-DV)**

21 Input:

22 — The parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e)$

23 — The sender's public identifier M_A

24 — The receiver's private key $K_B = (M_B + s)^{-1}Q_2$

25 — A ciphertext (c, S, T)

26 Assumptions: the parameters \mathcal{P} describe valid bilinear groups; R is a public key, an element of G_2 ; M_A and
27 M_B are both elements of Z_p ; K_B , an element of G_2 , is a valid generated value for M_B according to P-
28 DHI-GV; h is an element of Z_p and S is an element of G_1 .

29 Output:

1 — Either a rejection message or a pair made of a plaintext m and a valid signature (h, S) for m on
 2 behalf of the sender M_A .

3 Operation: Use the following steps.

4 1) Compute $u = e(T, K_B)$.

5 2) Compute $m = c \oplus H_3(u)$.

6 3) Compute $h = H_2(m, u)$.

7 4) Accept the message if $u = \frac{e(S, M_A Q_2 + R)}{e(Q_1, Q_2)^h}$.

8 5) If the test of step 4 is succeeds, output the plaintext m and the signature (h, S)

9 If both parties follow the specified algorithms, the sender always obtains the plaintext m together with a signature
 10 (h, S) bearing the name of the sender M_A .

11 11. Pairing-based Key Agreement Schemes

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

18 A key agreement scheme consists of a key agreement operation, along with supporting key management.
 19 Domain parameter and key pair generation for the key agreement schemes are specified further in
 20 Section?). A key agreement operation has the following form for all the schemes:

21 1) Establish one or more sets of valid domain parameters with which the parties' key pairs shall be
 22 associated.

23 2) Obtain one's valid private keys for the operation, associated with the domain parameters
 24 established in Step 1.

25 3) Establish one or more valid private keys for the operation, associated with the domain
 26 parameters established in Step 1

27 4) Obtain other party's identity based public key and one or more purported public keys for the
 28 operation.

29 5) (Optional.) Depending on the cryptographic operations in Step 5, choose an appropriate method
 30 to validate the public keys and the domain parameters. If any validation fails, output "invalid"
 31 and stop.

32 6) Apply certain cryptographic operations to the private and public keys to produce a shared secret
 33 value.

34 7) For each shared secret key to be agreed on, establish or agree on key derivation parameters and
 35 derive a shared secret key from the shared secret value and the key derivation parameters using a
 36 key derivation function.

1 NOTE 1—By the definition of a key agreement scheme, if the correct keys are used and computation is performed
 2 properly, the shared secret keys computed by the two parties will also be the same. However, to verify the identities of
 3 the parties and to ensure that they indeed possess the same key, the parties may need to perform a key confirmation
 4 protocol. See Annex D.5.1.3 of IEEE 1363-2000 for more details.

5 NOTE 2—A given public/private key pair may be used by either party for any number of key agreement operations,
 6 depending on the implementation.

7 NOTE 3—Depending on the key derivation function, there may be security-related constraints on the set of allowed
 8 key derivation parameters. The interpretation of these parameters is left to the implementation. For instance, it may
 9 contain key-specific information, protocol-related public information, and supplementary, private information. For
 10 security, the interpretation should be unambiguous. See Annex D.5.1.4 of IEEE 1363-2000 for further discussion.

11 NOTE 4—The attributes of the shared secret key depend on the particular key agreement scheme used, the attributes of
 12 the public/private key pairs, the nature of the parameters to the key derivation function, and whether or not key
 13 confirmation is performed. See Annex D.5.1 of IEEE 1363-2000 for further discussion of the attributes of the shared
 14 secret key.

15 NOTE 5—The two parties may produce errors under certain conditions, such as the following:

- 16 a. private key not found in step 2
- 17 b. public key not found in step 4
- 18 c. public key not valid in step 5
- 19 d. private key or public key not supported in step 6
- 20 e. key derivation parameter not supported in step 7

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

23 11.1 Wang key agreement

24 IBKAS-IDAK is Identity Based Key Agreement Scheme, IDAK version. Each party contributes two key
 25 pairs.

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

- 27 — A secret value derivation primitive, which shall be P-WKA-D1
- 28 — A key derivation function, which should be KDF1 (Section 13.1 of IEEE 1363-2000) or a function
 29 designated for use with IBKAS-IDAK in an addendum to this standard

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

32 11.1.1 Wang Key Agreement: Generate Shared Secrets (W-KA-G)

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

- 35 1) Obtain the valid set of elliptic curve domain parameters with which the parties' two identity
 36 based key pairs shall be associated.

- 1 2) Obtain the identity based private key w and select a valid key pair (u, v) for the operation,
2 associated with the parameters established in Step 1.
- 3 3) Obtain the other party's identity based public key w' and the purported public key v' for the
4 operation, associated with the parameters established in Step 1.
- 5 4) Compute a shared secret value z from the selected private keys w and u and the other party's two
6 public keys w' and v' with the selected secret value derivation primitive.
- 7 5) Convert the shared secret value z to an octet string Z using FE2OSP.
- 8 6) For each shared secret key to be agreed on:
- 9 i) Establish or otherwise agree on key derivation parameters P_i for the key.
- 10 ii) Derive a shared secret key K_i from the octet string Z and the key derivation
11 parameters P_i with the selected key derivation function (see Section 9.4.1).

12 A conformance region should include:

- 13 — At least one valid set of domain parameters
- 14 — At least one valid private key w for each set of domain parameters
- 15 — All valid key pairs (u, v) associated with the same set of domain parameters as w
- 16 — All valid public keys w' and v' associated with the same set of domain parameters as s ; if key
17 validation is performed, invalid public keys w' and v' that are appropriately handled by the
18 implementation may also be included in the conformance region
- 19 — A range of key derivation parameters P

20 11.2 SCC Key Agreement

21 Security of this scheme was proved in [ChenChengSmart07]. Each party contributes two key pairs.

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

- 23 — A secret value derivation primitive, which shall be P-SCK-D1
- 24 — A key derivation function, which should be KDF1 (Section 13.1 of IEEE 1363-2000) or a function
25 designated for use with SCC Key Agreement in an addendum to this standard

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

28 11.2.1 SCC Key Agreement: Generate Shared Secrets (SCC-KA-G)

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

- 31 1) Obtain the valid set of elliptic curve domain parameters with which the parties' two identity
32 based key pairs shall be associated.

- 1 2) Obtain the identity based private key d and select a valid key pair $(x, E = xP)$ [Should this
2 operation be addressed by using another primitive?] for the operation, associated with the
3 parameters established in Step 1.
- 4 3) Obtain the other party's identity based public key Q and the purported public key E' for the
5 operation, associated with the parameters established in Step 1.
- 6 4) Compute a shared secret value z from the selected private keys d and x and the other party's two
7 public keys Q and K_i with the selected secret value derivation primitive [this primitive hasn't
8 been specified].
- 9 5) Convert the shared secret value z to an octet string Z using FE2OSP.
- 10 6) For each shared secret key to be agreed on:
- 11 i) Establish or otherwise agree on key derivation parameters P_i for the key.
- 12 ii) Derive a shared secret key K_i from the octet string Z and the key derivation
13 parameters P_i with the selected key derivation function (see Section 9.4.1).

14 A conformance region should include:

- 15 — At least one valid set of domain parameters
- 16 — At least one valid private key d for each set of domain parameters
- 17 — All valid key pairs (x, E) associated with the same set of domain parameters as s
- 18 — All valid public keys Q and P_i associated with the same set of domain parameters as s ; if key
19 validation is performed, invalid public keys Q and P_i that are appropriately handled by the
20 implementation may also be included in the conformance region
- 21 — A range of key derivation parameters P

22 12. Pairing-based Proxy Re-encryption Schemes

23 A proxy re-encryption system allows the proxy to transform ciphertexts computed under an entity A's
24 public key into the different ciphertexts that can be decrypted by using the other entity B's secret key. This
25 system works as follows; the entity A or a trusted third party generates a re-encryption key and sets it in a
26 proxy. On receiving a ciphertext encrypted by A's public key, the proxy transforms the ciphertext by
27 running the re-encryption algorithm with the re-encryption key, and outputs the transformed ciphertext to
28 the entity B. B decrypts it by its secret key. This section denotes two types of proxy re-encryption schemes.

29 12.1 M1 Proxy Re-encryption Scheme

30 The scheme takes a security parameter t . It requires the definition of three hash functions:

- 31 — $H_1 : \{0, 1\}^* \rightarrow Z_p$, where $H_1(s) = IHF1(BS2OSP(s), p, t)$

32 The scheme also requires a random bit generation algorithm:

- 33 — R_1 : a source of random values in the set Z_p

1 In the following, each message is assumed to be elements of the group G_2 . It can be converted into an octet
2 string or a bit string using data type conversion described in section 5.6.

3 In this scheme, the proxy transforms a ciphertext computed by EV scheme manner into the different
4 ciphertext in BB2 scheme manner. The public key encryption scheme EV consists of the algorithms M1-
5 PRE-EX2, M1-PRE-EN2, M1-PRE-DE2, and the identity-based encryption scheme BB₂ consists of the
6 algorithms M1-PRE-S, M1-PRE-EX1, M1-PRE-EN1, and M1-PRE-DE1. The proxy has a re-encryption
7 key generated using M1-PRE-EX3 algorithm, and on receiving a ciphertext, the proxy transforms it into the
8 different ciphertext by running the algorithm M1-PRE-RE with the re-encryption key if the inputted
9 ciphertext is valid. The proxy verifies the ciphertext by running the algorithm M1-PRE-VE. The public
10 parameters in EV scheme are assumed to follow that in BB2 scheme. All algorithms in the proxy re-
11 encryption scheme are denoted in the following.

12 NOTE—It is recommended that applications establish a methodology for generating re-encryption key and
13 its deployment in the proxy. The details of those are beyond the scope of this document, but are critical for
14 the security of an implemented application. [M2007] describes an example how to generate and deploy the
15 re-encryption key.

16 12.1.1 M1 Proxy Re-encryption: Setup (M1-PRE-S)

17 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
18 steps. This operation creates a server secret which must be secured, as all secret keys calculated within the
19 system depend on it. It is recommended that applications establish a methodology for changing the server
20 secret and public parameters on a regular basis, and have a methodology for handling the disclosure of a
21 server secret.

22 The steps to setup the key server and system parameters are:

- 23 1) Establish the set of base groups and G_1, G_2 and a pairing $e : G_1 \times G_1 \rightarrow G_2$. Annex X contains
24 recommendations for the generation of these objects.
- 25 2) Select a random generator Q_1 in G_1 using A.X.X.
- 26 3) Select random elements Q_2 and Q_3 of G_1 .
- 27 4) Generate a random server secret s of Z_p .
- 28 5) Calculate the corresponding Q_4 as sQ_1 .
- 29 6) Calculate the pairing value V as $e(Q_2, Q_4)$.
- 30 7) Make the public parameters \mathcal{P} as the set $\mathcal{P} = (Q_1, Q_2, Q_3, Q_4, V, G_1, G_2, e)$ available. Secure
31 the server secrets s in a way appropriate to the application.

32 12.1.2 M1 Proxy Re-encryption: Extract1 (M1-PRE-EX1)

33 The extract operation takes an arbitrary identity string ID in $\{0, 1\}^*$ and calculates the corresponding secret
34 key elements. It is recommended that applications establish a methodology for authenticating access to
35 secret keys by using the ID string as an identity in a trusted authentication system. The details of
36 authenticating the key request are beyond the scope of this document, but are critical for the security of an
37 implemented application.

1 The steps to compute the secret key corresponding to an identity string ID are:

- 2 1) Compute the identity element $m = H_1(ID)$ of Z_p .
- 3 2) Using R_1 , generate a random integer u of Z_p .
- 4 3) Use the P-BB2-G primitive to compute $K_{0,m}$ and $K_{1,m}$ using m , u as the randomizer, the server
5 secret s , and the public parameters \mathcal{P} .
- 6 4) Output $\alpha_m = (K_{0,m}, K_{1,m})$ as the secret key.

7 **12.1.3 M1 Proxy Re-encryption: Encryption1 (M1-PRE-EN1)**

8 The encryption operation takes an arbitrary identity string ID in $\{0,1\}^*$, a message M of G_2 , and the
9 public parameters \mathcal{P} , and outputs a ciphertext to be transmitted to the receiver named by ID .

10 The steps to compute the ciphertext are:

- 11 1) Compute the identity element $m = H_1(ID)$ of Z_p .
- 12 2) Using R_1 , generate a random integer r of Z_p .
- 13 3) Use the P-BB2-E primitive to compute $E_{0,m}$, $E_{1,m}$ and B , using m , r as the message
14 randomizer, and the public parameters \mathcal{P} .
- 15 4) Compute $Y_m = MB$ of G_2 .
- 16 5) Output the triple $C_m = \langle E_{0,m}, E_{1,m}, Y_m \rangle$ as the ciphertext.

17 **12.1.4 M1 Proxy Re-encryption: Decryption1 (M1-PRE-DE1)**

18 The decryption operation takes a ciphertext C_m computed for identity ID where $m = H_1(ID)$ of Z_p , and
19 the secret key α_m that corresponds to m , and computes the message M that was encrypted by the sender.

20 The steps to compute the message M are:

- 21 1) Parse α_m as $(K_{0,m}, K_{1,m})$.
- 22 2) Parse C_m as $(E_{0,m}, E_{1,m}, Y_m)$.
- 23 3) Use the P-BB2-D primitive to compute B , using m , the secret key elements $K_{0,m}$ and $K_{1,m}$, and
24 the public parameters \mathcal{P} .
- 25 4) Output $M = Y_m / B$ of G_2 .

1 12.1.5 M1 Proxy Re-encryption: Extract2 (M1-PRE-EX2)

2 The extract operation takes the public parameters \mathcal{P} and calculates the secret key
3 elements k_0, k_1 and k_2 of Z_p , and the corresponding public key elements L_0, L_1 and L_2 of G_1 .

4 The steps to compute the secret key and the corresponding public key are:

- 5 1) Use the P-EV-G primitive to compute k_0, k_1, k_2, L_0, L_1 and L_2 using the public system
6 parameters \mathcal{P} .
- 7 2) Output $\beta = (k_0, k_1, k_2)$ as the secret key and $\gamma = (L_0, L_1, L_2)$ as the corresponding public key.

8 12.1.6 M1 Proxy Re-encryption: Encryption2 (M1-PRE-EN2)

9 The encryption operation takes a message M of G_2 , the public parameters \mathcal{P} for a key server, the public
10 key γ , outputs a ciphertext C to be transmitted to the receiver.

11 The steps to compute the ciphertext are:

- 12 1) Parse γ as (L_0, L_1, L_2) .
- 13 2) Using R_1 , generate a random integer r of Z_p .
- 14 3) Use the P-EV-E primitive to compute E_0, E_1, E_2 and B , using r as the message randomizer, the
15 public key elements L_0, L_1 and L_2 , and the public parameters \mathcal{P} .
- 16 4) Compute $Y = MB$ of G_2 .
- 17 5) Output the quadruple $C = (E_0, E_1, E_2, Y)$ as the ciphertext.

18 12.1.7 M1 Proxy Re-encryption: Decryption2 (M1-PRE-DE2)

19 The decrypt operation takes a ciphertext C and the secret key element k_2 according to P-EV-G and either
20 recovers the message M .

21 The steps to decrypt C into M are:

- 22 1) Parse $\beta = (k_0, k_1, k_2)$.
- 23 2) Parse C as (E_0, E_1, E_2, Y) .
- 24 3) Use the P-EV-D primitive to compute B , using E_2 as the ciphertext element, k_2 as the secret
25 key element, and the public parameters \mathcal{P} .
- 26 4) Compute $M = Y / B$ of G_2 .
- 27 5) Output M as the decrypted message.

28

1 12.1.8 M1 Proxy Re-encryption: Extract3 (M1-PRE-EX3)

2 The extract operation takes an arbitrary identity string ID in $\{0,1\}^*$, the public parameters \mathcal{P} , the secret key
3 element $K_{1,ID}$, the secret key β , and calculates the corresponding re-encryption key δ .

4 The steps to compute the re-encryption key corresponding to an identity string ID are:

- 5 1) Parse $\beta = (k_0, k_1, k_2)$.
- 6 2) Compute the identity element $m = H_1(ID)$ of Z_p .
- 7 3) Rename $K_{1,ID}$ to $K_{1,m}$.
- 8 4) Use the P-BR1-G primitive to compute u_0, u_1 and U using m , the secret key element $K_{1,m}$, the
9 secret key elements k_0, k_1 and k_2 , and the public system parameters \mathcal{P} .
- 10 5) Output $\delta = (u_0, u_1, U)$ as the re-encryption key.

11 12.1.9 M1 Proxy Re-encryption: Re-encryption (M1-PRE-RE)

12 The re-encryption operation takes an arbitrary identity string ID in $\{0,1\}^*$, the public parameters \mathcal{P} , the re-
13 encryption key δ , and a ciphertext C , outputs a re-encrypted ciphertext to be transmitted to the receiver.
14 Before re-encryption, the proxy verifies the ciphertext C and re-encrypts it if C is valid.

15 The steps to compute the ciphertext are:

- 16 1) Parse C as (E_0, E_1, E_2, Y) .
- 17 2) Run M1-PRE-VE to verify C , using the public key γ and the public parameters \mathcal{P} . If it is valid,
18 go to the next step. Otherwise output “error”.
- 19 3) Parse $\delta = (u_0, u_1, U)$.
- 20 4) Compute the identity element $m = H_1(ID)$ of Z_p .
- 21 5) Use the P-BR1-RE primitive to compute $E_{0,m}, E_{1,m}$ and R using m , the re-encryption key
22 elements u_0, u_1 and U , the ciphertext elements E_0, E_1 and E_2 , and the public parameters \mathcal{P} .
- 23 6) Compute $Y_m = RY$ of G_2 .
- 24 7) Output the quadruple $C_m = (E_{0,m}, E_{1,m}, Y_m)$ as the re-encrypted ciphertext.

25 12.1.10 M1 Proxy Re-encryption: Verification (M1-PRE-VE)

26 The verification operation takes the public parameters \mathcal{P} , the public key γ , and a ciphertext C , verify
27 the C .

- 1 The steps to verify the ciphertext C are:
- 2 1) Parse γ as (L_0, L_1, L_2) .
 - 3 2) Parse C as (E_0, E_1, E_2, Y) .
 - 4 3) Use the P-BR1-V primitive to verify C using the public system parameters \mathcal{P} , the ciphertext
 - 5 elements E_0, E_1 and E_2 , and the public key elements L_0, L_1 and L_2 .
 - 6 4) Return the output of P-BR1-V.

7 12.2 M2 Proxy Re-encryption scheme

8 In this scheme, the proxy transforms a ciphertext for an identity computed by the BB_2 scheme into a
 9 different ciphertext using the BB_2 scheme, but for another identity.. The proxy has a re-encryption key
 10 generated using M2-PRE-EX algorithm, and on receiving a ciphertext, the proxy transforms it into the
 11 different ciphertext by running the algorithm M2-PRE-RE with the re-encryption key if the inputted
 12 ciphertext is valid. The proxy verifies the ciphertext by running the algorithm M2-PRE-VE. These
 13 algorithms are denoted in the following.

14 NOTE—The algorithms for BB_2 scheme are described in the previous section. It is recommended that
 15 applications establish a methodology for generating re-encryption key and its deployment in the proxy. The
 16 details of those are beyond the scope of this document, but are critical for the security of an implemented
 17 application. [M2007] describes an example how to generate and deploy the re-encryption key.

18 12.2.1 M2 Proxy Re-encryption: Setup (M2-PRE-S)

19 The setup operation requires random generation of the parameters needed to operate the rest of the scheme
 20 steps. This operation creates a server secret which must be secured, as all secret keys calculated within the
 21 system depend on it. It is recommended that applications establish a methodology for changing the server
 22 secret and public parameters on a regular basis, and have a methodology for handling the disclosure of a
 23 server secret.

24 The steps to setup the key server and system parameters are:

- 25 1) Establish the set of base groups G_1, G_2 and a pairing $e : G_1 \times G_1 \rightarrow G_2$. Annex X contains
- 26 recommendations for the generation of these objects.
- 27 2) Select a random generator Q_1 of G_1 using A.X.X.
- 28 3) Select random elements Q_2 and Q_3 of G_1 .
- 29 4) Generate a random server secret s of Z_p .
- 30 5) Calculate the corresponding Q_4 as sQ_1 .
- 31 6) Calculate the pairing value V as $e(Q_2, Q_4)$.
- 32 7) Make the public parameters \mathcal{P} as the set $\mathcal{P} = (Q_1, Q_2, Q_3, Q_4, V, G_1, G_2, e)$ available. Secure
- 33 the server secrets s in a way appropriate to the application.

1 12.2.2 M2 Proxy Re-encryption: Extract (M2-PRE-EX)

2 The extract operation takes an arbitrary identity string ID_1 in $\{0,1\}^*$, the public parameters \mathcal{P} , the secret
3 key element K_{1,ID_1} associated with ID_1 , the server secret s , and calculates the corresponding re-encryption
4 key.

5 The steps to compute the re-encryption key corresponding to ID_1 are:

- 6 1) Compute the identity element $m_1 = H_1(ID_1)$ of Z_p .
- 7 2) Rename K_{1,ID_1} to K_{1,m_1} .
- 8 3) Use the P-BR2-G primitive to compute U using m_1 , the secret key element K_{1,m_1} , the server
9 secret s , and the public system parameters $\mathcal{P} = (Q_1, Q_2, Q_3, Q_4, V, G_1, G_2, e)$.
- 10 4) Output $U_{m_1} = U$ as the re-encryption key.

11 12.2.3 M2 Proxy Re-encryption: Re-encryption (M2-PRE-RE)

12 The re-encryption operation takes an arbitrary identity string ID_0 in $\{0,1\}^*$ and ID_1 in $\{0,1\}^*$, the public
13 parameters \mathcal{P} , the re-encryption key U_{m_1} , and a ciphertext C_{ID_0} , outputs a re-encrypted ciphertext C_{ID_1} to
14 be transmitted to the receiver ID_1 .

15 The steps to compute the ciphertext are:

- 16 1) Check if ID_0 delegates its decryption right to ID_1 . If so, then go to the next step. Otherwise
17 output “error”.
- 18 2) Compute the identity element $m_0 = H_1(ID_0)$ and $m_1 = H_1(ID_1)$ of Z_p .
- 19 3) Parse C_{ID_0} as $(E_{0,ID_0}, E_{1,ID_0}, Y_{ID_0})$ and rename those to $C_{m_0} = (E_{0,m_0}, E_{1,m_0}, Y_{m_0})$.
- 20 4) Run M2-PRE-VE to verify C_{m_0} . If it is valid then go to the next step. Otherwise output “error”.
- 21 5) Use the P-BR2-RE primitive to compute E_{0,m_1} , E_{1,m_1} and R_{m_1} using m_0 , m_1 , the re-encryption
22 key elements U_{m_1} , a ciphertext E_{0,m_0} and E_{1,m_0} , and the public system parameters \mathcal{P} .
- 23 6) Compute $Y_{m_1} = R_{m_1} Y_{m_0}$ of G_2 .
- 24 7) Output the quadruple $C_{m_1} = (E_{0,m_1}, E_{1,m_1}, Y_{m_1})$ as the re-encrypted ciphertext.

25 NOTE—It is out of scope describing how to check an entity delegates its decryption rights to the other in
26 step 1. For example, the proxy might have a table which is described all delegation associated to it and
27 checks the delegation referring to the table. In this case, the proxy has to manage the table securely so as
28 not to be accessed it by an unauthorized entity.

1 **12.2.4 M2 Proxy Re-encryption: Verification (M2-PRE-VE)**

2 The verification operation takes the public parameters \mathcal{P} , the public key element m , and a ciphertext C_m ,
3 verify the C_m .

4 The steps to verify the ciphertext C_m are:

- 5 1) Parse C_m as $(E_{0,m}, E_{1,m}, Y_m)$.
- 6 2) Use the P-BR2-V primitive to verify C_m using m , the public system parameters \mathcal{P} , the
7 ciphertext elements $E_{0,m}$ and $E_{1,m}$.
- 8 3) Return the output of P-BR2-V.

1 **Draft Standard for Identity-based**
2 **Public-key Cryptography Using**
3 **Pairings**

4