IEEE Standards

IEEE Std 1363a™-2004
(Amendment to
IEEE Std 1363™-2000)

1363a™

IEEE Standard Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques

IEEE Computer Society

Sponsored by the
Microprocessor and Microcomputer Standards Committee

IEEE

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

2 September 2004

Print: SH95223
PDF: SS95223

Recognized as an
American National Standard (ANSI)

IEEE Std 1363a™-2004
(Amendment to
IEEE Std 1363™-2000)

IEEE Standard Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques

Sponsor

Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society

Approved 22 July 2004

American National Standard Institute

Approved 25 March 2004

IEEE-SA Standards Board

Abstract: This standard specifies additional public-key cryptographic techniques beyond those in IEEE Std 1363-2000. It is intended to be merged with IEEE Std 1363-2000 during future revisions.
Keywords: digital signature, encryption, key agreement, public-key cryptography

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

Print: ISBN 0-7381-4003-1 SH95223
PDF: ISBN 0-7381-4004-X SS95223

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

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Secretary, IEEE-SA Standards Board

445 Hoes Lane

P.O. Box 1331

Piscataway, NJ 08855-1331USA

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

Introduction

(This introduction is not part of IEEE Std 1363a-2004, IEEE Standard for Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques.)

IEEE Std 1363a-2004 started in 1996 as an amendment to IEEE P1363, which later became IEEE Std 1363™-2000. At the time of the project's inception, the techniques included in P1363 were fairly stable; however, there were additional techniques that had been submitted to the Working Group that warranted further study and consideration. The P1363a project became the entry point for newly proposed and comparatively less-established techniques, while the P1363 document was solidified and prepared for the IEEE balloting process.

In early 1999, the Working Group began to limit the contents of P1363a primarily to include techniques that fit within the framework of the base document. In particular, the group concentrated on techniques in each family that performed functions that were not available in the P1363 document, such as encryption schemes in the discrete logarithm (DL) and elliptic curve (EC) families and signature schemes with message recovery in all families [DL, EC, and IF (integer factorization)].

Because of the large number of good candidates and the close similarities between many of the different techniques, the Working Group began to formalize the criteria for acceptance into P1363a, comparing all aspects of the techniques. By late 1999, the Working Group had converged on several candidate techniques to include in P1363a, making selections to provide a variety of relevant tradeoffs among security attributes, computational efficiency, and industry adoption for each family. As with the P1363 project, the Working Group chose to close P1363a to additional techniques and prepare it for ballot, leaving standardization of additional techniques that may not have been ready for inclusion at that time to future standards.

Throughout the process of creating IEEE Std 1363a-2004, the Working Group has been very fortunate to receive numerous submissions of creative, useful, and well-designed public-key cryptographic techniques. In addition to techniques in the IF, DL, and EC families that were not included in 1363a, there were also a number of techniques submitted that fell outside of the scope of both IEEE Std 1363-2000 and IEEE Std 1363a-2004. As a result, the Microprocessor Standards Committee (MSC) commissioned a study group, in which many of the 1363 working group members participated, to explore the possibility of creating additional standards related to public-key cryptography. In late 2000, the working group began work on P1363.1 (Public-Key Cryptographic Techniques Based on Hard Problems over Lattices) and P1363.2 (Password-Based Public-Key Cryptographic Techniques). At the time of this writing, the Working Group is also attempting to put together a project for the second amendment to Std 1363-2000 (P1363b), which would continue the work from IEEE Std 1363-2000 and IEEE Std 1363a-2004. The Working Group is considering the creation of a registry for public-key cryptographic techniques.

The process of developing IEEE Std 1363a-2004 has been very challenging, but it has allowed the Working Group to continue to refine the general understanding of public-key cryptography and to follow up on the work put into IEEE Std 1363-2000. The IEEE P1363 Working Group continues to be an excellent forum for experts to discuss technical and standardization issues associated with public-key cryptography. It has provided a focal point for the presentation of new developments in public-key cryptography and remains a source for up-to-date information on the topic. For the duration of its existence, the Working Group intends to maintain a web page that will support all of the 1363 standards and current projects (see http://grouper.ieee.org/groups/1363/index.html).

Notice to users

Errata

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

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates, terms, and conditions of the license agreements offered by patent holders or patent applicants. Further information may be obtained from the IEEE Standards Department.

Organization

This standard is written as an amendment to the IEEE Std 1363-2000 document and is intended to be merged into a future version of that document. Familiarity with IEEE Std 1363-2000 is assumed. The following clauses and annexes are updated:

- Clause 2, References
- Clause 3, Definitions
- Clause 4, Types of cryptographic techniques
- Clause 5, Mathematical conventions
- Clause 6, Primitives based on the discrete logarithm problem
- Clause 7, Primitives based on the elliptic curve discrete logarithm problem
- Clause 8, Primitives based on the integer factorization problem
- Clause 9, Key agreement schemes
- Clause 10, Signature schemes
- Clause 11, Encryption schemes
- Clause 12, Message-encoding methods
- Clause 13, Key derivation functions
- Clause 14, Auxiliary techniques (new title)
- Annex A (informative) Number-theoretic algorithms
- Annex B (normative) Conformance
- Annex C (informative) Rationale

— Annex D (informative) Security considerations

— Annex E (informative) Formats

— Annex G (informative) Bibliography (new number; was Annex F)

One new annex is inserted:

— Annex F (informative) Information about patents

Participants

The following is a list of participants in the IEEE P1363 Working Group.

William Whyte, *Chair* **Don Johnson**, *Vice-Chair*
Ari Singer, *Secretary* **David Jablon**, *Treasurer*
Mike Brenner, *Primary Editor* **Burt Kaliski**, *P1363a Technical Editor*

Beni Arazi
Daniel V. Bailey
Daniel Brown
Lily Chen
Wei Dai
Louis Finkelstein
Gurgen Kachatryan
Pieter Kasselman
Tetsutaro Kobayashi
David Kravitz
Pil Joong Lee
Daniel Lieman
Michael Markowitz
Tatsuaki Okamoto
Satomi Okazaki
Anand Rajan
Allen Roginsky
Roger Schlafly
Rich Schroeppel
Jerry Solinas
David Stern

In addition, the Working Group would like to thank the following people for their contributions to this amendment, whether by submitting techniques for consideration, participating in mailing list discussions, or attending meetings.

Michel Abdalla
Tolga Acar
Carlisle Adams
Kochira Akiyama
Kazumato Aoki
Terry Arnold
Paulo Barreto
Mihir Bellare
Simon Blake-Wilson
Daniel Bleichenbacher
Jurjen Bos
Dong Hyun Cheon
Bram Cohen
Nicolas Courtois
Carlin Covey
Scott Crenshaw
Lucien Dancanet
Lei Fang
Ludovic Flament
Eiichiro Fujisaki
Walter Fumy
Ernst Giessmann
Tom Gindin
Louis Goubin
Peter Gutmann
Safuat Hamdy
Florian Hess
Shouichi Hirose
Robert Hofer
Erik Falk Højsted
Seak Hie Hong
David Hopwood
Fumitaka Hoshino
Jim Hughes
Jakob Jonsson
KCDSA Task Force Team
Chang Han Kim
Hee Jin Kim
Jeong-Soo Kim
Kunio Kobayashi
Kristin Lauter
Peter Lawrence
Byoungcheon Lee
Chang-Hyi Lee
Sung Jae Lee
Franck Leprevost
Jon In Lim
Helger Lipmaa
Moses Liskov
Stefan Löwe
Phil MacKenzie
John Malone-Lee
James Manger
Marcel Martin
Stephen M. Matyas
Preda Mihailescu
Bodo Möller
Jean Monnerat
Shiho Moriai
Hikaru Morita
James Muir
Anthony Mulcahy
Hong Nguyen
Sang Ho Oh
Hilarie Orman
Christof Paar
China Pellacuru
Mohammad Peyravian
David Pointcheval
Martin Rehwald
Holger Reif
Leo Reyzin
Phillip Rogaway
Rei Safavi-Naini
Erkay Savas
Dominikus Scherkl
Martin Schulze
Thomas Schweinberger
Michael Scott
Victor Shoup
Joe Silverman
Nigel Smart
David Sowinski
Kazuo Takaragi
Scott Vanstone
Serge Vaudenay
Tom Wu
Ok Yeon Yi
Yiqun Lisa Yin
Joong Chul Yoon
Susumu Yoshida
Yuliang Zheng

The Working Group apologizes for any inadvertent omissions from the previous list. Please note that inclusion of a person's name on the previous two lists on page v does not imply that the person agrees with all of the materials in this amendment.

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

Terry Arnold
Steven Bard
Mitchell Bonnett
Guru Dutt Dhingra
Dante Del Corso
Sourav Dutta
Neil Horman
Don Johnson
Jack Johnson
Burt Kaliski
David Kravitz
Thomas M. Kurihara
Gregory Luri
Robert Mortonson
David M. Rockwell
Tom Schaal
Roger Schlafly
Ari Singer
Joseph Tardo
Jerry Thrasher
Joan Viaplana
Paul Work
Don Wright
Cheng-Wen Wu
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 25 March 2004, it had the following membership:

Don Wright, *Chair*
Steve M. Mills, *Vice Chair*
Judith Gorman, *Secretary*

Chuck Adams
H. Stephen Berger
Mark D. Bowman
Joseph A. Bruder
Bob Davis
Roberto de Boisson
Julian Forster*
Arnold M. Greenspan
Mark S. Halpin
Raymond Hapeman
Richard J. Holleman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Hermann Koch
Thomas J. McGean
Daleep C. Mohla
Paul Nikolich
T. W. Olsen
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Doug Topping
Joe D. Watson

*Member Emeritus

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

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Don Messina
IEEE Standards Project Editor

Contents

1. Overview 2

1.1 Scope 2
1.2 Purpose 2

2. References 2

3. Definitions 3

4. Types of cryptographic techniques 4

4.2 Primitives 4
4.3 Schemes 4
4.4 Additional methods 4
4.5 Table summary 5

5. Mathematical conventions 7

5.1 Mathematical notation 7
5.4 Elliptic curves and points 9
5.5 Data type conversion 10

6. Primitives based on the discrete logarithm problem 15

6.2 Primitives 15

7. Primitives based on the elliptic curve discrete logarithm problem 21

7.2 Primitives 21

8. Primitives based on the integer factorization problem 26

8.2 Primitives 28

9. Key agreement schemes 33

9.2 DL/ECKAS-DH1 33
9.3 DL/ECKAS-DH2 34
9.4 DL/ECKAS-MQV 35

10. Signature schemes 36

10.1 General model 36
10.2 DL/ECSSA 38
10.3 IFSSA 39
10.4 DL/ECSSR 41
10.5 DL/ECSSR-PV 43
10.6 IFSSR 46

11. Encryption schemes 48

11.1 General model 48
11.2 IFES 50
11.3 DL/ECIES 50
11.4 IFES-EPOC 55

12. Message-encoding methods 57

12.1 Message-encoding methods for signatures with appendix 57
12.2 Message-encoding methods for encryption 64
12.3 Message-encoding methods for signatures giving message recovery 71

13. Key derivation functions 77

13.2 KDF2 78

14. Auxiliary techniques 79

14.1 Hash functions 79
14.2 Mask generation functions 82
14.3 Symmetric encryption schemes 83
14.4 Message authentication codes 87

Annex A (informative) Number-theoretic algorithms 89

Annex B (normative) Conformance 111

Annex C (informative) Rationale 114

Annex D (informative) Security considerations 119

Annex E (informative) Formats 147

Annex F (informative) Informative about patents 151

Annex G (informative) Bibliography 152

IEEE Standard Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques

EDITOR'S NOTE—The editing instructions contained in this amendment define how to merge the material contained herein into the existing base standard and its amendments to form the comprehensive standard.

The editing instructions are shown in ***bold italic***. Four editing instructions are used: change, delete, insert, and replace. ***Change*** is used to make small corrections in existing text or tables. The editing instruction specifies the location of the change and describes what is being changed either by using ~~strikethrough~~ (to remove old material) or underscore (to add new material). ***Delete*** removes existing material. ***Insert*** adds new material without disturbing the existing material. Insertions may require renumbering. If so, renumbering instructions are given in the editing instructions. ***Replace*** is used to make large changes in existing text, subclauses, tables, or figures by removing existing material and replacing it with new material. Editorial notes will not be carried over into future editions.

1. Overview

Replace subclauses 1.1 and 1.2 with the following text:

1.1 Scope

Specifications of common public-key cryptographic techniques supplemental to those considered in IEEE Std 1363™-2000, including mathematical primitives for secret value (key) derivation, public-key encryption, digital signatures, and identification, and cryptographic schemes based on those primitives, are provided. Specifications of related cryptographic parameters, public keys and private keys, are also provided. Class of computer and communications systems is not restricted.

1.2 Purpose

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

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

When the IEEE 1363a project began, work was in progress in IEEE Std 1363-2000 on specifying many of the basic techniques in a standard way. Additional techniques remained to be specified, which could eventually be added to IEEE Std 1363-2000. However, many of those additional techniques required further development before being standardized, whereas the basic techniques in IEEE Std 1363-2000 were relatively more established. To facilitate the completion of the work on the basic techniques while also providing a forum for discussing additional techniques, the IEEE 1363 Working Group sought to have separate projects for the two efforts, which would result in separate, companion standards for some period of time. It is the working group's intention that the standards will be merged during future revisions.

Replace Clause 2 with the following text:

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision shall apply.

ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation.[1]

ANSI X9.71-2000, Keyed Hash Message Authentication Code (MAC).

FIPS PUB 46-3, Data Encryption Standard (DES), Federal Information Processing Standards Publication 46-3, U.S. Department of Commerce/National Institute of Standards and Technology, October 1999.[2]

[1]ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

FIPS PUB 81, DES Modes of Operation, Federal Information Processing Standards Publication 81, U.S. Department of Commerce/National Institute of Standards and Technology, December 1980.

FIPS PUB 180-2, Secure Hash Standard, Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce/National Institute of Standards and Technology, August 1, 2002.

FIPS PUB 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, U.S. Department of Commerce/National Institute of Standards and Technology, November 2001.

FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198, U.S. Department of Commerce/National Institute of Standards and Technology, March 2002.

ISO/IEC 10118-3:1998, Information Technology-Security Techniques—Hash-Functions—Part 3: Dedicated hash-functions.[3]

NOTES

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

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

3—A bibliography is provided in Annex G.[4]

3. Definitions

Replace 3.37 with the following text:

3.37 private key: The private element of the public/private key pair. (Note that for convenience of implementation, some components of the public key may be included in the private key as well.) *See also:* **public/private key pair**; **valid key**.

Delete 3.50, insert the following subclause after 3.48, and renumber the existing 3.49 as 3.50:

3.49 signature scheme giving message recovery: A digital signature scheme where the signature contains enough information for recovery of part or all of the message that is signed. *Contrast:* **signature scheme with appendix**. *See also:* **digital signature**; **digital signature scheme**.

Insert the following definitions in alphabetical order and renumber as required:

binary field: A finite field containing 2^m elements for some integer $m > 0$. Also known as characteristic two finite field.

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

[3]ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

[4]Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

characteristic: The prime integer p for the finite field $GF(p^m)$.

odd-characteristic extension field: A finite field containing p^m elements for some integer $m > 1$ where p is an odd prime number.

NOTE—Although mathematically, the various techniques based on odd-characteristic extension fields may generally be employed when $m = 1$, for the purposes of this standard, it will be assumed that $m > 1$ when the term "odd-characteristic extension field" is given.

prime field: A finite field of p elements, where p is an odd prime number. Also known as prime finite field.

4. Types of cryptographic techniques

4.2 Primitives

Replace the list of types of primitives with the following text:

— Secret Value Derivation Primitives (SVDP): components of key agreement schemes
— Pre-Signature Primitives (PSP), Signature Primitives (SP), and Verification Primitives (VP): components of signature schemes
— Encryption Primitives (EP) and Decryption Primitives (DP): components of encryption schemes

4.3 Schemes

Replace the list of types of schemes with the following text:

— Key agreement schemes (KAS), in which two parties use their public/private key pairs, and possibly other information to agree on a shared secret key. The shared secret key may then be used for symmetric cryptography. (See Clause 3 for the definition of symmetric cryptography.)
— Signature schemes with appendix (SSA), in which one party signs a message using its private key, and any other party can verify the signature by examining the message, the signature, and the signer's corresponding public key.
— Signature schemes giving (message) recovery (SSR), in which one party signs a message using its private key such that part or all of the message can be recovered from the signature, and any other party can verify the signature, and recover that part of the message by examining the nonrecoverable part of the message (if any), the signature and the signer's corresponding public key.
— Encryption schemes (ES), in which any party can encrypt a message using a recipient's public key, and only the recipient can decrypt the message by using its corresponding private key. Encryption schemes may be used for establishing secret keys to be used in symmetric cryptography.

Replace 4.4 with the following text:

4.4 Additional methods

This standard specifies the following additional methods:

— Message-encoding methods, which are components of signature or encryption schemes:
  1) Encoding methods for signatures with appendix (EMSA)
  2) Encoding methods for signatures giving message recovery (EMSR)
  3) Encoding methods for encryption (EME)

— Key derivation functions (KDF), which are components of key agreement schemes
— Auxiliary techniques, which are building blocks for other additional methods
  1) Mask generation functions (MGF), which are used in EME
  2) Hash functions, which are used in EMSA, EMSR, EME, KDF, and MGF
  3) Symmetric encryption schemes, which are used in some encryption schemes, EMSR and EME
  4) Message authentication codes (MAC), which are a component of some encryption schemes

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

Replace 4.5 with the following text:

4.5 Table summary

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

Table 1—Summary of techniques in the standard

Scheme name	Primitives	Additional methods
Key agreement schemes		
DL/ECKAS-DH1	DL/ECSVDP-DH or -DHC	KDF1 or KDF2 (each uses a hash function)
DL/ECKAS-DH2	DL/ECSVDP-DH and/or -DHC	KDF1 or KDF2 (each uses a hash function)
DL/ECKAS-MQV	DL/ECSVDP-MQV or -MQVC	KDF1 or KDF2 (each uses a hash function)
Signature schemes with appendix		
DL/ECSSA	DLSP-NR and DLVP-NR OR DLSP-DSA and DLVP-DSA OR ECSP-NR and ECVP-NR OR ECSP-DSA and ECVP-DSA	EMSA1 (uses a hash function)
IFSSA	IFSP-RSA1 and IFVP-RSA1 OR IFSP-RSA2 and IFVP-RSA2 OR IFSP-RW and IFVP-RW OR IFSP-ESIGN and IFVP-ESIGN	EMSA2 or EMSA3 or EMSA4 or EMSA5 (each uses a hash function; EMSA4 and EMSA5 also use a mask generation function)
Signature schemes giving message recovery		

Table 1—Summary of techniques in the standard *(continued)*

Scheme name	Primitives	Additional methods
DL/ECSSR	DLPSP-NR2/PV, DLSP-NR2 and DLVP-NR2 OR ECPSP-NR2/PV, ECSP-NR2 and ECVP-NR2	EMSR1 (uses a hash function)
DL/ECSSR-PV	DLPSP-NR2/PV, DLSP-PV and DLVP-PV OR ECPSP-NR2/PV, ECSP-PV and ECVP-PV	EMSR2 (uses a key derivation function and possibly a symmetric encryption scheme); a hash function
IFSSR	IFSP-RSA1 and IFVP-RSA1 OR IFSP-RSA2 and IFVP-RSA2 OR IFSP-RW and IFVP-RW	EMSR3 (uses a hash function and a mask generation function)
Encryption schemes		
DL/ECIES	DL/ECSVDP-DH or -DHC	KDF2; MAC1 (each uses a hash function); possibly a symmetric encryption scheme
IFES	IFEP-RSA and IFDP-RSA	EME1 (uses a hash function and a mask generation function)
IFES-EPOC	IFEP-OU and IFDP-OU	EME2 or EME3 (each uses a hash function and a mask generation function; EME3 also uses a key derivation function and possibly a symmetric encryption scheme)

NOTE—Acronym definitions are as follows:

Families—DL: discrete logarithm; EC: elliptic curve; IF: integer factorization

Schemes—KAS: key agreement scheme; SSA: signature scheme with appendix; SSR: signature scheme giving message recovery; IES: integrated encryption scheme; ES: encryption scheme

Primitives—SVDP: secret value derivation primitive; PSP: pre-signature primitive; SP: signature primitive; VP: verification primitive; EP: encryption primitive; DP: decryption primitive

Additional methods—KDF: key derivation function; EMSA: encoding method for signatures with appendix; EMSR: encoding method for signatures giving message recovery; EME: encoding method for encryption; MAC: message authentication code

Names of techniques—DH: Diffie-Hellman; DHC: Diffie-Hellman with cofactor exponentiation/multiplication; MQV: Menezes-Qu-Vanstone; MQVC: Menezes-Qu-Vanstone with cofactor exponentiation/multiplication; NR: Nyberg-RueppelDSA: Digital signature algorithm; RSA: Rivest-Shamir-Adleman; RW: Rabin-Williams; PV: Pintsov-Vanstone; OU: Okamoto-Uchiyama

5. Mathematical conventions

5.1 Mathematical notation

Replace the definition of GF(q) with the following text:

GF(*q*) The finite field containing q elements. For this standard, *q* will be either a power of an odd prime *p* or a power of 2.

Insert the following definition:

GF(p^m) The finite field containing pm elements for some integer $m > 1$ where *p* is an odd prime number. Also known as odd-characteristic extension field. See 5.3.3 for more information.

NOTE—Although mathematically, the various techniques based on *GF*(p^m) may generally also be employed when $m = 1$, for the purposes of this standard, it will be assumed that $m > 1$ when the notation "*GF*(p^m)" or the term "odd-characteristic extension field" is given. The cases *GF*(p^m) and *GF*(*p*) are thus treated separately.

Insert the following subclause after 5.3.2.2:

5.3.2.3 Composite basis over *GF*(2^d)

This representation is possible for fields *GF*(2^m) if *m* is a composite integer. The representation is determined by choosing a subfield *GF*(2^d) of *GF*(2^m), where $1 < d < m$, and $m = ds$. Elements of *GF*(2^m) are then represented as vectors of *s* elements from *GF*(2^d) using any basis representation, including those in 5.3.2.3.1 or 5.3.2.3.2. Elements of *GF*(2^d) may in turn be represented in any basis, including those in 5.3.2.1 or 5.3.2.2, or with another composite basis if *d* is a composite integer. This process of choosing representations for subfields of *GF*(2^d) may be repeated for each of the divisors of *d*.

5.3.2.3.1 Polynomial basis over *GF*(2^d)

This representation is similar to that defined in 5.3.2.1 with the difference that $d \geq 1$. This representation is determined by choosing an irreducible polynomial *f(t)* over *GF*(2^d) of degree $s = m/d$. Then *GF*(2^m) is isomorphic to *GF*(2^d)*[t]/f(t)*.

Then if the polynomial basis representation over *GF*(2^d) is used, for purposes of conversion, the string

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

where each a_i is a bit string of length *d* shall be taken to represent the polynomial

$$a_{s-1}t^{s-1} + \ldots + a_2 t^2 + a_1 t + a_0$$

where the coefficients a_i are elements of *GF*(2^d). The coefficients a_i will in turn be represented with some basis over some subfield of *GF*(2^d) [perhaps *GF*(2)].

As in 5.3.2.1, the additive identity (zero) element of the field *GF*(2^m) is represented by a string of *s* representations of the zero element in the chosen representation for *GF*(2^d). Thus, if *GF*(2^d) is represented with a polynomial or normal basis, the zero element of *GF*(2^m) is represented by a string of all 0 bits. Similarly, the multiplicative identity (one) element of *GF*(2^m) is represented by a string of $(s - 1)$ representations of the zero element of *GF*(2^d) followed by the representation of the one element of *GF*(2^d). Thus, if *GF*(2^d) is rep-

resented with a polynomial basis, the one element of $GF(2^m)$ is represented by a string of $m - 1$, 0 bits followed by a 1-bit. If $GF(2^d)$ is represented by a normal basis, the one element of $GF(2^m)$ is represented by a string of $(s - 1)d$ 0-bits followed by d 1-bits.

The arithmetic in this basis is identical to that of 5.3.2.1 with the exception that all coefficient operations are performed in the field $GF(2^d)$.

5.3.2.3.2 Normal basis over $GF(2^d)$

This representation is similar to that defined in 5.3.2.2 with the difference that $d \geq 1$. This representation is determined by choosing a normal polynomial *f(t)* of degree $s = m/d$. Then $GF(2^m)$ is isomorphic to $GF(2^d)[t]/f(t)$.

Then if the normal basis representation over $GF(2^d)$ is used, for purposes of conversion, the string

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

where each a_i is a bit string of length d, shall be taken to represent the element

$$a_0\theta + a_1\theta^2 + a_1\theta^{2^2} + \ldots + a_{s-1}\theta^{2^{s-1(2)}}$$

where q is a root of *f(t)* in $GF(2^s)$. The coefficients a_i will in turn be represented with some basis over some subfield of $GF(2^d)$ [perhaps $GF(2)$].

As in 5.3.2.2, the additive identity (zero) element of the field $GF(2^m)$ is represented by a string of s representations of zero in $GF(2^d)$. For example, if $GF(2^d)$ is represented with a polynomial or normal basis, a string of 0 bits represents 0 in $GF(2^m)$. The multiplicative identity (one) element of the field is represented by a string of s representations of one in $GF(2^d)$. For example, if $GF(2^d)$ is represented with a polynomial basis, the string consists of the pattern: $(d - 1)$ 0 bits followed by a 1-bit. The pattern repeated s times represents one in $GF(2^m)$. If $GF(2^d)$ is represented with a normal basis, the one element of $GF(2^m)$ is represented with a string of all 1-bits.

The arithmetic in this basis is identical to that of 5.3.2.1 with the exception that all coefficient operations are performed in the field $GF(2^d)$.

Insert the following subclause after 5.3.2:

5.3.3 Odd-characteristic extension field

An *odd-characteristic extension field* is a finite field whose number of elements is a power of an odd prime. If $m > 1$, then there is a unique field (up to isomorphism) $GF(p^m)$ with p^m elements. For the purposes of conversion, the elements of $GF(p^m)$ shall be represented in a polynomial basis and converted to integers in the set $\{0, 1, \ldots, p^m - 1\}$ as follows. The polynomial basis representation is determined by choosing an irreducible polynomial *f(t)* over $GF(p)$. Then $GF(p^m)$ is isomorphic to $GF(p)[t]/f(t)$. The element with representation

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

shall be represented by the integer

$$a_{m-1}p^{m-1} + \ldots + a_2p^2 + a_1p + a_0$$

(note that each a^i is between 0 and $p - 1$, so the result is between 0 and $p^m - 1$). (The coefficients can be recovered from this integer by successively dividing by p and keeping the remainder. For instance, denoting the integer by a, the coefficient a_0 can be obtained as $a_0 = a \bmod p$, then a can be updated to $a = \lfloor a/p \rfloor$, then a_1 can be obtained as $a_1 = a \bmod p$, and so on.)

Note that for the purposes of conversion, the additive identity (zero) element of the field is represented by the integer 0, and the multiplicative identity (one) element of the field is represented by the integer 1.

A description of the arithmetic of $GF(p^m)$ is given in A.17.

5.4 Elliptic curves and points

Replace the second and third paragraphs and equations with the following text:

If $q = p^m$, $p > 3$, $m \geq 1$, then a and b shall satisfy $4a^3 + 27b^2 \neq 0$ in $GF(q)$, and every point $P = (x_P, y_P)$ on E (other than the point O) shall satisfy the following equation in $GF(q)$:

$$y_p^2 = x_p^3 + ax_p + b$$

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

$$y_p^2 + x_p y_p = x_p^3 + ax_p^2 + b$$

If q is a power of 3, then a and b shall be nonzero in $GF(q)$, and every point $P = (x_P, y_P)$ on E (other than the point O) shall satisfy the following equation in $GF(q)$:

$$y_p^2 = x_p^3 + ax_p^2 + b$$

5.5 Data type conversion

Replace Figure 1 with the following:

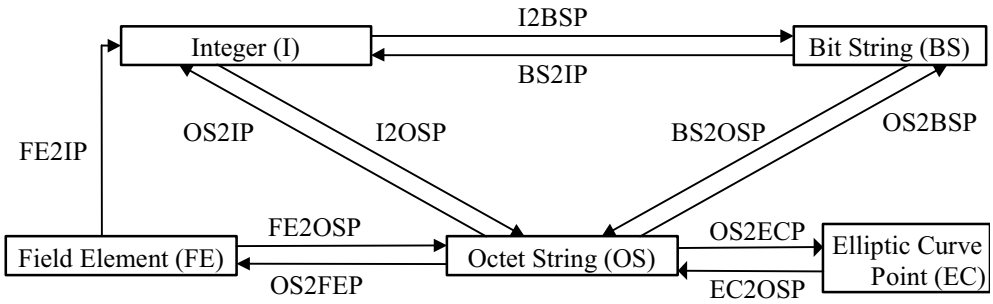

Figure 1—Summary of data type conversion primitives

Replace 5.5.4 with the following text:

5.5.4 Converting between finite field elements and octet strings (FE2OSP and OS2FEP)

An element x of a finite field $GF(q)$, for the purposes of conversion, is represented by an integer if q is an odd prime or an odd prime power (see 5.3.1 and 5.3.3) or by a bit string if q is a power of 2 (see 5.3.2). If q is an odd prime or an odd prime power, then to represent x as an octet string, I2OSP shall be used with the integer value representing x and the length $\lceil \log_{256} q \rceil$ as inputs. If q is a power of 2, then to represent x as an octet string, BS2OSP shall be applied to the bit string representing x.

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

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

Insert the following subclause after 5.5.5:

5.5.6 Converting between elliptic curve points and octet strings

An elliptic curve point P (which is not the point at infinity O) can be represented in either compressed or uncompressed form. (For internal calculations, it may be advantageous to use other representations, e.g., the projective coordinates of A.9.6. Also see A.9.6 for more information on point compression.) The uncompressed form of P is simply given by its two coordinates. The compressed form is presented in 5.5.6.1. The octet string format is defined to support both compressed and uncompressed points.

5.5.6.1 Compressed elliptic curve points

The *compressed form* of an elliptic curve point $P \neq O$ defined over $GF(p)$ and $GF(2^m)$ is the pair $(x_P, \tilde{y}_p)$ where x_P is the x-coordinate of P, and $\tilde{y}_p$ is a bit that is computed as defined next.

Two compressed forms are defined: an LSB *compressed form* defined for elliptic curves over $GF(p)$ and $GF(2^m)$, and an SORT *compressed form* defined for elliptic curves over $GF(2^m)$ and $GF(p^m)$.

NOTE—Elliptic curve points over $GF(2^m)$ can be compressed with either the LSB or SORT method. The LSB method is equivalent to the point compression method in Annex E of IEEE Std 1363-2000.

5.5.6.1.1 LSB compressed form

Over $GF(p)$, the LSB compressed form has $\tilde{y}_p$ = FE2IP (y_P) mod 2. Put another way, $\tilde{y}_p$ is the rightmost bit of y_P.

Over $GF(2^m)$, the LSB compressed form has $\tilde{y}_p = 0$ if $x_P = 0$ and $\tilde{y}_p$ = FE2IP ($y_P x_P^{-1}$) mod 2 if $x_P \neq 0$. That is, $\tilde{y}_p$ is the rightmost bit of the field element $y_P x_P^{-1}$ if $x_P \neq 0$. (This rule applies for any of the basis representations given in 5.3.2.)

Procedures for *point decompression* (i.e., recovering y_P given x_P and $\tilde{y}_p$) are given in A.12.8 (for q an odd prime) and A.12.9 (for q a power of 2).

NOTE—The name “LSB” refers to the fact that the compressed bit is the least significant bit of the integer representation of y_P or $y_P x_P^{-1}$.

5.5.6.1.2 SORT compressed form

Let (x_P, y'_P) be the inverse of the point (x_P, y_P). (For $GF(p^m)$, $y'_P = -y_P$; for $GF(2^m)$, $y'_P = x_P + y_P$).

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

A procedure for point decompression is given in A.12.11.

NOTES

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

2—Although the SORT compressed form is defined here for any field, in the representations in 5.5.6.2, it is only employed for elliptic curves over $GF(2^m)$ and $GF(p^m)$.

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

5.5.6.2 Two-coordinate point representations

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

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

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

$$PO = PC \| X \| Y$$

where

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

— Bit S is 1 if the format uses the SORT compressed form; 0 otherwise.
— Bit U is 1 if the format is uncompressed or hybrid; 0 otherwise.
— Bit C is 1 if the format is compressed or hybrid; 0 otherwise.
— Bit $\tilde{Y}$ is equal to the bit $\tilde{y}_p$ if the format is compressed or hybrid; 0 otherwise.

X is the octet string of length $\lceil \log_{256} q \rceil$ representing x_P according to FE2OSP (see 5.5.4).

Y is the octet string of length $\lceil \log_{256} q \rceil$ representing y_P of P according to FE2OSP (see 5.5.4) if the format is uncompressed or hybrid; Y is an empty string if the format is compressed.

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

The primitive that converts octet strings to elliptic curve points is called Octet String to Elliptic Curve Point Conversion Primitive-*R*, or OS2ECP-*R*. It takes the octet string and the field size q as inputs and outputs the corresponding elliptic curve point, or "error." It shall use OS2FEP to get x_P. It shall use OS2FEP to get y_P if the format is uncompressed, and it may output "error" if the recovered point is not on the elliptic 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 either of these two means if the format is hybrid, and if the format is hybrid, it may output "error" if different values are obtained by the two means. It shall output "error" in the following cases:

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

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

5.5.6.2.1 Uncompressed representation: EC2OSP-XY and OS2ECP-XY

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

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

5.5.6.2.2 LSB compressed representation: EC2OSP-XL and OS2ECP-XL

This representation is defined for elliptic curves over $GF(p)$ and $GF(2^m)$ only.

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 LSB compressed form, the octet string *X* shall represent x_P, and the octet string *Y* shall be the empty string. The length of the octet string *PO* shall be $1 + \lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSP-XL and OS2ECP-XL.

5.5.6.2.3 SORT compressed representation: EC2OSP-XS and OS2ECP-XS

This representation is defined for elliptic curves over $GF(2^m)$ and $GF(p^m)$ only.

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

5.5.6.2.4 LSB hybrid representation: EC2OSP-XYL and OS2ECP-XYL

This representation is defined for elliptic curves over $GF(p)$ and $GF(2^m)$ only.

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

5.5.6.2.5 SORT hybrid representation: EC2OSP-XYS and OS2ECP-XYS

This representation is defined for elliptic curves over $GF(2^m)$ and $GF(p^m)$ only.

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 SORT compressed form, and the octet strings *X* and *Y* shall represent x_P and y_P, respectively. The length of the octet string PO shall be 1+ 2 $\lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSP-XYS and OS2ECP-XYS.

5.5.6.3 *x*-coordinate only representation: EC2OSP-X and OS2ECP-X

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

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

In this representation, the octet *PC* shall have binary value 0000 0001, the octet string *X* shall represent x_P, and the octet string *Y* shall be empty. The length of the octet string *PO* shall be 1+ $\lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSP-X and OSC2ECP-X.

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

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

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

5.5.6.4 Summary of representations

Table 2 summarizes the point representations in 5.5.6.2 and 5.5.6.3.

Table 2—Elliptic curve point representations

Representation	Primitives	PC	X	Y	Finite fields
Uncompressed	EC2OSP-XY and OS2ECP-XY	0000 0100	x_P	y_P	All
LSB compressed	EC2OSP-XL and OS2ECP-XL	0000 001$\tilde{Y}$	x_P	Empty	$GF(p)$ and $GF(2^m)$
SORT compressed	EC2OSP-XS and OS2ECP-XS	0000 011$\tilde{Y}$	x_P	Empty	$GF(2^m)$ and $GF(p^m)$
LSB hybrid	EC2OSP-XYL and OS2ECP-XYL	0000 011$\tilde{Y}$	x_P	y_P	$GF(p)$ and $GF(2^m)$
SORT hybrid	EC2OSP-XYS and OS2ECP-XYS	0000 111$\tilde{Y}$	x_P	y_P	$GF(2^m)$ and $GF(p^m)$
x-coordinate-only	EC2OSP-X and OS2ECP-X	0000 0001	x_P	Empty	All
Point O	All	0000 0000	Empty	Empty	All

NOTES

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

2—The various representations employ distinct values for the first octet PC, so the octet strings produced by the different representations are nonoverlapping, except at the point O. Consequently, a “generic” OS2ECP primitive may be constructed that handles all of the representations.

6. Primitives based on the discrete logarithm problem

6.1.1 Notations

Insert the following entries in the list of notation:

(c, d)	Signature, a pair of integers, computed by a signature primitive
c	Signature part, an integer, computed by a signature primitive
d	Signature part, an integer, computed by a signature primitive
h	Randomized hash value, an integer
i	Pre-signature, an integer
u	Randomizer, an integer
C	Encrypted message, an octet string, computed by an encryption scheme
M_1	Recoverable message part, an octet string, the part of a message that can be recovered from the signature in a signature scheme giving message recovery
M_2	Nonrecoverable message part (if any), an octet string, the part of a message that cannot be recovered from the signature in a signature scheme giving message recovery
T	Authentication tag, a bit string, computed by an encryption scheme

6.1.2 DL domain parameters

Replace the first paragraph with the following text:

DL *domain parameters* are used in every DL primitive and scheme and are an implicit component of every DL key. A set of DL domain parameters specifies a field $GF(q)$, where q is a positive odd prime integer p, 2^m for some positive integer m, or p^m for an odd prime p and some integer $m \geq 2$; a positive prime integer r dividing $(q - 1)$; and a field element g of multiplicative order r (g is called the *generator* of the subgroup of order r). If $q = 2^m$, it also specifies a representation for the elements of $GF(q)$ to be used by the conversion primitives (see 5.3 and 5.5). Implicitly it also specifies the cofactor $k = (q - 1) / r$. If the DLSVDP-DHC or DLSVDP-MQVC primitive is to be applied, then it shall also be the case that GCD $(k, r) = 1$ (i.e., r does not divide k; see A.1.1).

6.2 Primitives

6.2.5 DLSP-NR

Replace the first paragraph with the following text:

DLSP-NR is the Discrete Logarithm Signature Primitive, Nyberg-Rueppel version. It is based on the work of Nyberg and Rueppel [B120].[5] It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the DLVP-NR primitive. Note, however, that the DLPSP-NR2/PV and DLSP-NR2 primitives are intended for signature schemes giving message recovery in this version of the standard. DLSP-NR can be invoked in the scheme DLSSA as part of signature generation.

[5]The numbers in brackets correspond to those in the bibliography in Annex G.

Replace the note with the following text:

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated and stored by the signer following the security recommendations of D.3.1, D.4.1.2, D.6, and D.7. The private key should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new key pair should be generated for every signature. The one-time private key *u* should be securely deleted after step 4), as its recovery by an opponent can lead to the recovery of the private key *s*.

2—The one-time key pair in step 1) may be precomputed in advance of the availability of the message representative. See D.5.2.1 for further discussion.

6.2.6 DLVP-NR

Replace the first paragraph with the following text:

DLVP-NR is the Discrete Logarithm Verification Primitive, Nyberg–Rueppel version. It is based on the work of Nyberg and Rueppel [B120]. This primitive recovers the message representative that was signed with DLSP-NR, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that the DLVP-NR2 primitive is intended for signature schemes giving message recovery in this version of the standard. DLVP-NR can be invoked in the scheme DLSSA as part of signature verification.

6.2.7 DLSP-DSA

Replace the note with the following text:

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated and stored by the signer following the security recommendations of D.3.1, D.4.1.2, D.6, and D.7. The private key should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new key pair should be generated for every signature. The one-time private key *u* should be securely deleted after step 4), as its recovery by an opponent can lead to the recovery of the private key *s*.

2—Similar to DLSP-NR, the one-time key pair in step 1) may be precomputed in advance of the availability of the message representative. Step 2) and step 3) as well as the values u^{-1} and *sc* in step 4) may also be precomputed. See D.5.2.1 for further discussion.

6.2.8 DLVP-DSA

Insert the following five subclauses (6.2.9–6.2.13) after 6.2.8:

6.2.9 DLPSP-NR2/PV

DLPSP-NR2/PV is the Discrete Logarithm Pre-Signature Primitive, Nyberg–Rueppel/Pintsov–Vanstone version. It is based on the work of ISO/IEC 9796-3:2000 [B79], Nyberg and Rueppel [B120], and Pintsov and Vanstone [PV99]. The primitive generates a randomizer and a pre-signature for a signature scheme. It can be invoked in the scheme DLSSR or DLSSR-PV as part of signature generation. DLPSP-NR2/PV corresponds to the first two steps of DLSP-NR.

Input: The DL domain parameters *q*, *r,* and *g*

Assumptions: DL domain parameters *q*, *r,* and *g* are valid.

Output:

— The randomizer, which is an integer u such that $1 \le u < r$
— The pre-signature, which is an integer i such that $1 \le i < q$

Operation: The randomizer u and the pre-signature i shall be computed by the following or an equivalent sequence of steps:

1) Generate a key pair (u, v) with the set of domain parameters. (See Notes 1 and 2 below.)
2) Convert v into an integer i with primitive FE2IP [recall that v is an element of $GF(q)$].
3) Output u as the randomizer and i as the pre-signature.

Conformance region recommendation: A conformance region should include:

— At least one valid set of DL domain parameters q, r, and g

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated by the signer following the security recommendations of D.3.1, D.4.1.2, D.6, and D.7. The randomizer should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new randomizer/pre-signature pair should be generated for every signature. (If signatures are computed with the same randomizer for two or more different message representatives, or if the randomizer becomes known, an opponent may be able to recover the signer's private key.) The randomizer should be stored following the same security recommendations as for an ephemeral private key (it should be securely deleted after it is used in DLSP-NR2 or DLSP-PV).

2—Similar to DLSP-NR, the key pair may be precomputed in advance of the availability of the message representative.

6.2.10 DLSP-NR2

DLSP-NR2 is the Discrete Logarithm Signature Primitive, Nyberg–Rueppel version 2. It is based on the work of ISO/IEC 9796-3:2000 [B79] and Nyberg and Rueppel [B120]. The primitive generates a signature on a message representative with the private key of the signer, given a randomizer and a pre-signature generated by DLPSP-NR2/PV, in such a way that the message representative and the pre-signature can be recovered from the signature using the public key of the signer by the DLVP-NR or DLVP-NR2 primitive. It can be invoked in the scheme DLSSR as part of signature generation. DLSP-NR2 corresponds to the last three steps of DLSP-NR.

Except for having DL domain parameters as input rather than EC domain parameters, the primitive is identical to ECSP-NR2.

Input:

— The DL domain parameters q, r, and g associated with the key s
— The signer's private key s
— The randomizer, which is an integer u such that $1 \le u < r$
— The pre-signature, which is an integer i such that $1 \le i < q$
— The message representative, which is an integer f such that $0 \le f < r$

Assumptions: Private key s and DL domain parameters q, r, and g are valid and associated with each other; $1 \le u < r$; $1 \le i < q$; $0 \le f < r$.

Output: The signature, which is a pair of integers (c, d) such that $1 \le c < r$ and $0 \le d < r$; or "error"

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

1) Compute an integer $c = i + f \bmod r$. If $c = 0$, then output "error." (This is an extremely rare event for recommended values of r—see Note 2.)
2) Compute an integer $d = u – sc \bmod r$.
3) Output the pair (c, d) as the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid set of DL domain parameters q, r, and g
— At least one valid private key s for each set of domain parameters
— All randomizers u in the range [1, $r – 1$]
— All pre-signatures i in the range [1, $q – 1$]
— All message representatives f in the range [0, $r – 1$]

NOTES

1—Following the discussion for DLPSP-NR2/PV in this subclause, for security reasons, the randomizer u should be securely deleted after step 2).

2—The probability that the primitive will output "error" is about $1/r$ (i.e., essentially 0 for recommended values of r), assuming the pre-signature i is generated at random by DLPSP-NR2/PV. Thus, the check at the end of step 1) is included for completeness only; it is unlikely to occur in practice. The motivation for the check is to ensure that the signature depends on the signer's private key s; otherwise the signature could be verified with anyone's public key with the same domain parameters. [An "error" output in any case does not disclose any information about the private key s, because the private key is not involved in the pre-signature primitive or prior to step 2).]

6.2.11 DLVP-NR2

DLVP-NR2 is the Discrete Logarithm Verification Primitive, Nyberg–Rueppel version 2. It is based on the work of ISO/IEC 9796-3:2000 [B79] and Nyberg and Rueppel [B120]. This primitive recovers the message representative that was signed with DLSP-NR or DLSP-NR2, given only the signature and public key of the signer, and also recovers the pre-signature. It can be invoked in the scheme DLSSR as part of signature verification and message recovery. DLVP-NR2 is the same as DLVP-NR, except that the recovered pre-signature is output in addition to the message representative.

Input:

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

Assumptions: Public key w and DL domain parameters q, r, and g are valid and associated with each other.

Output:

— The message representative, which is an integer f such that $0 \le f < r$; or "invalid"
— The pre-signature, which is an integer i such that $1 \le i < q$

Operation: The message representative *f* and the pre-signature *i* shall be computed by the following or an equivalent sequence of steps:

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

Conformance region recommendation: A conformance region should include:

— At least one valid set of DL domain parameters *q*, *r*, and *g*
— At least one valid public key *w* for each set of domain parameters
— All purported signatures (*c*, *d*) that can be input to the implementation; this should at least include all (*c*, *d*) such that c and d are in the range [0, *r* – 1]

6.2.12 DLSP-PV

DLSP-PV is the Discrete Logarithm Signature Primitive, Pintsov–Vanstone version. It is based on the work of Pintsov and Vanstone [PV99]. The primitive generates a signature part on a randomized hash value with the private key of the signer, given a randomizer generated by DLPSP-NR2/PV, in such a way that the pre-signature generated by DLPSP-NR2/PV can be recovered from the signature part using the public key of the signer by the DLVP-PV primitive. It can be invoked in the scheme DLSSR-PV as part of signature generation.

Except for having DL domain parameters as input rather than EC domain parameters, the primitive is identical to ECSP-PV.

Input:

— The DL domain parameters *q*, *r*, and *g* associated with the key *s*
— The signer's private key *s*
— The randomizer, which is an integer *u* such that $1 \le u < r$
— The randomized hash value, which is a non-negative integer *h*

Assumptions: Private key *s* and DL domain parameters *q*, *r*, and *g* are valid and associated with each other; $1 \le u < r$; $h \ge 0$.

Output: A signature part, which is an integer d, where $0 \le d < r$

Operation: The signature part *d* shall be computed by the following or an equivalent sequence of steps:

1) Compute an integer $d = u - sh \bmod r$.
2) Output *d* as the signature part.

Conformance region recommendation: A conformance region should include:

— At least one valid set of DL domain parameters *q*, *r*, and *g*
— At least one valid private key *s* for each set of domain parameters
— All randomizers *u* in the range [1, *r* – 1]
— All randomized hash values *h* in the range [0, *r* – 1], where *r* is from the domain parameters of *s*

NOTES

1—Following the discussion for DLPSP-NR2/PV in this subclause, for security reasons, the randomizer *u* should be securely deleted after step 1).

2—This primitive does not check whether $h = 0$ to ensure that the signature depends on the signer's public key, because it is such an unlikely occurrence in the scheme in which DLSP-PV is employed, but an implementation could do so if desired.

6.2.13 DLVP-PV

DLVP-PV is the Discrete Logarithm Verification Primitive, Pintsov–Vanstone version. It is based on the work of Pintsov and Vanstone [PV99]. This primitive recovers a pre-signature from a signature part generated with DLSP-PV, given only the randomized hash value and the public key of the signer. It can be invoked in the scheme DLSSR-PV as part of signature verification and message recovery.

Input:

- The DL domain parameters *q*, *r,* and *g* associated with the key *w*
- The signer's public key *w*
- The randomized hash value, which is an integer $h \geq 0$
- The signature part from which the pre-signature is to be recovered, which an integer *d*

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

Output: The pre-signature, which is an integer *i* such that $1 \leq i < q$

Operation: The pre-signature *i* shall be computed by the following or an equivalent sequence of steps:

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

Conformance region recommendation: A conformance region should include:

- At least one valid set of DL domain parameters *q*, *r*, and *g*
- At least one valid public key w for each set of domain parameters
- All randomized hash values $h \geq 0$
- All signature parts *d* that can be input to the implementation; this should at least include all *d* such that d is in the range $[0, r - 1]$

7. Primitives based on the elliptic curve discrete logarithm problem

7.1.1 Notations

Insert the following entries in the list of notation:

(c, d)	Signature, a pair of integers, computed by a signature primitive
c	Signature part, an integer, computed by a signature primitive
d	Signature part, an integer, computed by a signature primitive
h	Randomized hash value, an integer
i	Pre-signature, an integer
u	Randomizer, an integer
C	Encrypted message, an octet string, computed by an encryption scheme
M_1	Recoverable message part, an octet string, the part of a message that can be recovered from the signature in a signature scheme giving message recovery
M_2	Nonrecoverable message part (if any), an octet string, the part of a message that cannot be recovered from the signature in a signature scheme giving message recovery
T	Authentication tag, a bit string, computed by an encryption scheme

7.1.2 EC domain parameters

EC domain parameters are used in every EC primitive and scheme and are an implicit component of every EC key. A set of EC domain parameters specifies a field $GF(q)$, where q is a positive odd prime integer p, 2^m for some positive integer m, or p^m for an odd prime p and some integer $m \geq 2$; two elliptic curve coefficients a and b, elements of $GF(q)$, that define an elliptic curve E; a positive prime integer r dividing the number of points on E; and a curve point G of order r (G is called the generator of a subgroup of order r). If $q = 2^m$, it also specifies a representation for the elements of $GF(q)$ to be used by the conversion primitives (see 5.3 and 5.5). Implicitly it also specifies the cofactor $k = \#E/r$ (where $\#E$ is the number of points on E). If key validation is to be performed or if the ECSVDP-DHC or ECSVDP-MQVC primitive is to be applied, then it shall also be the case that GCD $(k, r) = 1$ (i.e., r does not divide k; see A.1.1).

7.2 Primitives

7.2.5 ECSP-NR

Replace the first paragraph with the following text:

ECSP-NR is the Elliptic Curve Signature Primitive, Nyberg–Rueppel version. It is based on the work of Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the ECVP-NR primitive. Note, however, that the ECPSP-NR2/PV and ECSP-NR2 primitives are intended for signature schemes giving message recovery in this version of the standard. ECSP-NR can be invoked in the scheme ECSSA as part of signature generation.

Replace the note with the following text:

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated and stored by the signer following the security recommendations of D.3.1, D.4.2.2, D.6, and D.7. The private key should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new key pair should be generated for every signature. The one-time private key *u* should be securely deleted after step 4), as its recovery by an opponent can lead to the recovery of the private key *s*.

2—The one-time key pair in step 1) may be precomputed in advance of the availability of the message representative. See D.5.2.1 for further discussion.

7.2.6 ECVP-NR

Replace the first paragraph with the following text:

ECVP-NR is the Elliptic Curve Verification Primitive, Nyberg–Rueppel version. It is based on the work of Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. This primitive recovers the message representative that was signed with ECSP-NR, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. Note, however, that the ECVP-NR2 primitive is intended for signature schemes giving message recovery in this version of the standard. ECVP-NR can be invoked in the scheme ECSSA as part of signature verification.

7.2.7 ECSP-DSA

Replace the note with the following text:

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated and stored by the signer following the security recommendations of D.3.1, D.4.2.2, D.6, and D.7. The private key should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new key pair should be generated for every signature. The one-time private key *u* should be securely deleted after step 4), as its recovery by an opponent can lead to the recovery of the private key *s*.

2—Similar to ECSP-NR, the one-time key pair in step 1) may be precomputed in advance of the availability of the message representative. Step 2) and step 3) as well as the values u^{-1} and *sc* in step 4) may also be precomputed. See D.5.2.1 for further discussion.

Insert the following five subclauses (7.2.9–7.2.13) after 7.2.8:

7.2.9 ECPSP-NR2/PV

ECPSP-NR2/PV is the Elliptic Curve Pre-Signature Primitive, Nyberg–Rueppel and Pintsov–Vanstone version. It is based on the work of ISO/IEC 9796-3:2000 [B79], Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. The primitive generates a randomizer and a pre-signature for a signature scheme. It can be invoked in the scheme ECSSR or ECSSR-PV as part of signature generation. ECPSP-NR2/PV corresponds to the first two steps of ECSP-NR.

Input: The EC domain parameters *q*, *a*, *b*, *r*, and *G*.

Assumptions: EC domain parameters *q*, *a*, *b*, *r*, and *G* are valid.

Output:

— The randomizer, which is an integer u such that $1 \le u < r$
— The pre-signature, which is an integer i such that $1 \le i < q$

Operation: The randomizer u and the pre-signature i shall be computed by the following or an equivalent sequence of steps:

1) Generate a key pair (u, V) with the set of domain parameters. (See Notes 1 and 2 below.) Let $V = (x_V, y_V)$ ($V \neq O$ because V is a public key).
2) Convert x_V into an integer i with primitive FE2IP [recall that x_V is an element of $GF(q)$].
3) Output u as the randomizer and i as the pre-signature.

Conformance region recommendation. A conformance region should include:

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

NOTES

1—The key pair in step 1) should be a one-time key pair that is generated by the signer following the security recommendations of D.3.1, D.4.1.2, D.6, and D.7. The randomizer should be selected in an unbiased manner (see D.5.2.1 for further discussion), and a new randomizer/pre-signature pair should be generated for every signature. (If signatures are computed with the same randomizer for two or more different message representatives, or if the randomizer becomes known, an opponent may be able to recover the signer's private key.) The randomizer should be stored following the same security recommendations as for an ephemeral private key (it should be securely deleted after it is used in ECSP-NR2 or ECSP-PV).

2—Similar to ECSP-NR, the key pair may be precomputed in advance of the availability of the message representative.

7.2.10 ECSP-NR2

ECSP-NR2 is the Elliptic Curve Signature Primitive, Nyberg–Rueppel version 2. It is based on the work of ISO/IEC 9796-3:2000 [B79], Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. The primitive generates a signature on a message representative with the private key of the signer, given a randomizer and a pre-signature generated by ECPSP-NR2/PV, in such a way that the message representative and the pre-signature can be recovered from the signature using the public key of the signer by the ECVP-NR or ECVP-NR2 primitive. It can be invoked in the scheme ECSSR as part of signature generation. ECSP-NR2 corresponds to the last three steps of ECSP-NR.

Except for having EC domain parameters as input rather than DL domain parameters, the primitive is identical to DLSP-NR.

Input:

— The EC domain parameters q, a, b, r, and G associated with the key s
— The signer's private key s
— The randomizer, which is an integer u such that $1 \le u < r$
— The pre-signature, which is an integer i such that $1 \le i < q$
— The message representative, which is an integer f such that $0 \le f < r$

Assumptions: Private key s and EC domain parameters q, a, b, r, and G are valid and associated with each other; $1 \le u < r$; $1 \le i < q$; $0 \le f < r$.

Output: The signature, which is a pair of integers (c, d) such that $0 \le c < r$ and $0 \le d < r$ or "error"

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

1) Compute an integer $c = i + f \bmod r$. If $c = 0$, then output "error." (This is an extremely rare event for recommended values of r—see Note 2.)
2) Compute an integer $d = u - sc \bmod r$.
3) Output the pair (c, d) as the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid set of EC domain parameters q, a, b, r, and G
— At least one valid private key s for each set of domain parameters
— All randomizers u in the range $[1, r - 1]$
— All pre-signatures i in the range $[1, q - 1]$
— All message representatives f in the range $[0, r - 1]$

NOTES

1—Following the discussion for ECPSP-NR2/PV in this subclause, for security reasons, the randomizer u should be securely deleted after step 2).

2—The probability that the primitive will output "error" is about $1/r$ (i.e., essentially 0 for recommended values of r), assuming the pre-signature i is generated at random by ECPSP-NR2/PV. Thus, the check at the end of step 1) is included for completeness only; it is unlikely to occur in practice. The motivation for the check is to ensure that the signature depends on the signer's private key s; otherwise the signature could be verified with anyone's public key with the same domain parameters. [An "error" output in any case does not disclose any information about the private key s, because the private key is not involved in the pre-signature primitive or prior to step 2).]

7.2.11 ECVP-NR2

ECVP-NR is the Elliptic Curve Verification Primitive, Nyberg–Rueppel version 2. It is based on the work of ISO/IEC 9796-3:2000 [B79], Koblitz [B94], Miller [B117], and Nyberg and Rueppel [B120]. This primitive recovers the message representative that was signed with ECSP-NR2, given only the signature and public key of the signer, and recovers the pre-signature. It can be invoked in the scheme ECSSR as part of signature verification and message recovery. ECVP-NR2 is the same as ECVP-NR, except that the recovered pre-signature is output in addition to the message representative.

Input:

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

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

Output:

— The message representative, which is an integer f such that $0 \le f < r$; or "invalid"
— The pre-signature, which is an integer i such that $1 \le i < q$

Operation: The message representative *f* and the pre-signature *i* shall be computed by the following or an equivalent sequence of steps:

1) If *c* is not in [1, *r* – 1] or *d* is not in [0, *r* – 1], output "invalid" and stop.
2) Compute an elliptic curve point $P = d\,G + c\,W$. If P = O, output "invalid" and stop. Otherwise, $P = (x_P, y_P)$.
3) Convert the field element x_P to an integer *i* with primitive FE2IP.
4) Compute an integer $f = c - i \bmod r$.
5) Output *f* as the message representative and *i* as the pre-signature.

Conformance region recommendation: A conformance region should include:

— At least one valid set of EC domain parameters *q*, *a*, *b*, *r*, and *G*
— At least one valid public key *W* for each set of domain parameters
— All purported signatures (*c*, *d*) that can be input to the implementation; this should at least include all (*c*, *d*) such that *c* and *d* are in the range [0, *r* – 1]

7.2.12 ECSP-PV

ECSP-PV is the Elliptic Curve Signature Primitive, Pintsov–Vanstone version. It is based on the work of Koblitz [B94], Miller [B117], and Pintsov and Vanstone [PV99]. The primitive generates a signature part on a randomized hash value with the private key of the signer, given a randomizer generated by ECPSP-NR2/PV, in such a way that the pre-signature generated by ECPSP-NR2/PV can be recovered from the signature using the public key of the signer by the ECVP-PV primitive. It can be invoked in the scheme ECSSR-PV as part of signature generation.

Except for having EC domain parameters as input rather than DL domain parameters, the primitive is identical to DLSP-PV.

Input:

— The EC domain parameters *q*, *a*, *b*, *r*, and *G* associated with the key *s*
— The signer's private key *s*
— The randomizer, which is an integer *u* such that $1 \le f < r$
— The randomized hash value, which is a nonnegative integer *h*

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

Output: A signature part, which is an integer *d*, where $0 \le d < r$

Operation: The signature part d shall be computed by the following or an equivalent sequence of steps:

1) Compute an integer $d = u - sh \bmod r$.
2) Output the pair *d* as the signature part.

Conformance region recommendation: A conformance region should include:

— At least one valid set of EC domain parameters *q*, *a*, *b*, *r*, and *G*
— At least one valid private key *s* for each set of domain parameters
— All randomizers *u* in the range [1, *r* – 1]
— All randomized hash values *h* in the range [0, *r* – 1]

NOTES

1—For security reasons, the randomizer *u* should be securely deleted after step 2).

2—This primitive does not check whether $h = 0$ to ensure that the signature depends on the signer's public key, because it is such an unlikely occurrence in the scheme in which ECSP-PV is employed, but an implementation could do if desired.

7.2.13 ECVP-PV

ECVP-PV is the Elliptic Curve Verification Primitive, Pintsov–Vanstone version. It is based on the work of Koblitz [B94], Miller [B117], and Pintsov and Vanstone [PV99]. This primitive recovers a pre-signature from a signature part generated with ECSP-PV, given only the randomized hash value and the public key of the signer. It can be invoked in the scheme ECSSR-PV as part of signature verification and message recovery.

Input:

- The EC domain parameters q, a, b, r, and G associated with the key W
- The signer's public key W
- The randomized hash value, which is an integer $h \geq 0$
- The signature part from which the pre-signature is to be recovered, which is an integer d

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

Output: The pre-signature, which is an integer i such that $1 \leq i < q$

Operation: The pre-signature i shall be computed by the following or an equivalent sequence of steps:

1) If d is not in $[0, r - 1]$, output "invalid" and stop.
2) Compute an elliptic curve point $P = d\,G + h\,W$. If $P = O$, output "invalid" and stop. Otherwise, $P = (x_P, y_P)$.
3) Convert the field element x_P to an integer i with primitive FE2IP.
4) Output i as the pre-signature.

Conformance region recommendation: A conformance region should include:

- At least one valid set of EC domain parameters q, a, b, r, and G
- At least one valid public key W for each set of domain parameters
- All randomized hash values $h \geq 0$
- All signature parts d that can be input to the implementation; this should at least include all d such that d is in the range $[0, r - 1]$

8. Primitives based on the integer factorization problem

Replace the second paragraph with the following text:

There are four types of primitives in this family. Although they are all related to the integer factorization problem, they are based on different keys and operations. One type is known as RSA, for "Rivest-Shamir-Adleman" (see Rivest et al. [B129]); a second is known as RW, for "Rabin–Williams" (see Rabin [B128] and Williams [B149]). The third is known as OU, for "Okamoto–Uchiyama" (see Okamoto and Uchiyama [OU98]), and the fourth is known as ESIGN (see Fujisaki and Okamoto [FO98] and Okamoto [Oka90]).

8.1.1 Notation

Replace the following entries in the list of notation:

p, q	The prime factors of the modulus *n*
e	The public exponent, part of a public key (*e* is odd for RSA; *e* is even for RW; $e \geq 8$ for ESIGN)

Insert the following entries:

k	The length in bits of the primes for OU and ESIGN
u	Part of the OU public key
u_p	The value $\exp(u, p-1) \bmod p^2$, used in OU
r	Randomized hash value, an integer
v	Part of the OU public key
L(x)	The function $(x-1)/p$, used in OU
(n, u, v, k)	An OU public key
(p, q, k, w)	An OU private key
(n, e, k)	An ESIGN public key
(p, q, k, e)	An ESIGN private key
C	Encrypted message, an octet string, computed by an encryption scheme
M_1	Recoverable message part, an octet string, the part of a message that can be recovered from the signature in a signature scheme giving message recovery
M_2	Nonrecoverable message part (if any), an octet string, the part of a message that cannot be recovered from the signature in a signature scheme giving message recovery

Insert the following two subclauses (8.1.3.3 and 8.1.3.4) after 8.1.3.2:

8.1.3.3 OU key pairs

An OU *public key* consists of a *modulus n*, which is of the form p^2q for two odd positive prime integers *p* and *q*, where the length of each of *p* and *q* is *k* bits for some integer *k*, a pair of integers (u, v) $(1 < u < n, 1 < v < n)$ such that the order of $u_p = \exp(u, p-1) \bmod p^2$ is *p* and where $v = v_0^n \bmod n$ for some integer v_0 $(1 < v_0 < n)$. Define the function $L(x)$ as

$$L(x) = (x-1)/p$$

and let $w = L(u_p)$. The corresponding OU private key consists of the primes *p* and *q*, the length parameter *k*, and the value *w*.

NOTE—The value v_0 need not be kept secret. A typical selection of the value v_0 is *u*. See D.5.3.2.1 for further discussion.

8.1.3.4 ESIGN key pairs

An ESIGN *public key* consists of a *modulus n*, which is of the form p^2q for two odd positive prime integers *p* and *q* where the length of each of *p* and *q* is *k* bits for some integer *k* and the length of *n* is 3*k* bits, a *public exponent e* $(8 \leq e < n)$ such that GCD $(e, n) = 1$, and the length parameter *k*. The corresponding ESIGN *private key* consists of the pair of primes (p, q), the length parameter *k*, and the exponent *e*.

Replace the existing 8.1.3.3 with the following text (note the renumbering):

8.1.3.5 Considerations common to IF key pairs

An IF key pair may or may not be generated by the party using it, depending on the trust model. This and other security issues related to IF key pair generation are discussed in D.3.1 and D.4.3. A suggested method for generating RSA key pairs is contained in A.16.11. A suggested method for generating RW key pairs is contained in A.16.12.

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

Key validation is the process of determining whether a key is valid. Further discussion of key validation is contained in D.3.3, although no algorithm for IF public key validation is suggested in Annex A. No algorithm is given for IF private key validation, because, generally, a party controls its own private key and need not validate it. However, private key validation may be performed if desired.

For security, the primes *p* and *q* need to be generated so that they are unpredictable and inaccessible to an adversary. Whatever form a private key is represented in, it needs to be stored so that it is inaccessible to an adversary. The compromise of the primes or other private key components (that are not already part of the public key) may aid the adversary in recovering the private key. These values should be securely deleted if not used.

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

8.2 Primitives

Replace the third paragraph with the following text:

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

8.2.4 IFSP-RSA1

Replace the first paragraph with the following text:

IFSP-RSA1 is the RSA Signature Primitive, version 1. It is based on the work of Rivest et al. [B129]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RSA1 primitive. IFSP-RSA1 can be invoked in the schemes IFSSA and IFSSR as part of signature generation.

8.2.5 IFVP-RSA1

Replace the first paragraph with the following text:

IFVP-RSA1 is the RSA Verification Primitive, version 1. It is based on the work of Rivest et al. [B129]. This primitive recovers the message representative that was signed with IFSP-RSA1, given only the signature and public key of the signer. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. IFVP-RSA1 can be invoked in the schemes IFSSA and IFSSR as part of signature verification.

8.2.6 IFSP-RSA2

Replace the first paragraph with the following text:

IFSP-RSA2 is the RSA Signature Primitive, version 2. It is based on the work of ISO/IEC 9796:1991 [B78] and Rivest et al. [B129]. Its output is at least 1 bit shorter than the RSA modulus *n*. It can be invoked in a scheme to compute a signature on a message representative of a certain form with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RSA2 primitive. IFSP-RSA2 can be invoked in the schemes IFSSA and IFSSR as part of signature generation.

8.2.7 IFVP-RSA2

Replace the first paragraph with the following text:

IFVP-RSA2 is the RSA Verification Primitive, version 2. It is based on the work of ISO/IEC 9796:1991 [B78] and Rivest et al. [B129]. This primitive recovers the message representative that was signed with IFSP-RSA2, given only the signature, the public key of the signer, and the last four bits of the message representative. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. IFVP-RSA2 can be invoked in the schemes IFSSA and IFSSR as part of signature verification.

8.2.8 IFSP-RW

Replace the first paragraph with the following text:

IFSP-RW is the RW Signature Primitive. It is based on the work of ISO/IEC 9796:1991 [B78], Rabin [B128], and Williams [B149]. Its output is at least 1 bit shorter than the RW modulus *n*. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-RW primitive. IFSP-RW can be invoked in the schemes IFSSA and IFSSR as part of signature generation.

8.2.9 IFVP-RW

Replace the first paragraph with the following text:

IFVP-RW is the RW Verification Primitive. It is based on the work of ISO/IEC 9796:1991 [B78], Rabin [B128], and Williams [B149]. This primitive recovers the message representative that was signed with IFSP-RW, given only the signature, public key of the signer, and the last four bits of the message representative. It can be invoked in a scheme as part of signature verification and, possibly, message recovery. IFVP-RW can be invoked in the schemes IFSSA and IFSSR as part of signature verification.

Insert the following four subclauses (8.2.10–8.2.13) after 8.2.9:

8.2.10 IFEP-OU

IFEP-OU is the OU Encryption Primitive. It is based on the work of Okamoto and Uchiyama [OU98]. It is invoked in the scheme IFES-EPOC as part of encrypting a message, given the message and the public key of the intended recipient, and it is also used to verify that decryption was successful. The message can be decrypted within a scheme by invoking IFDP-OU.

Input:

— The recipient's OU public key (n, u, v, k)
— The message representative, which is an integer f such that $0 \le f < 2^{k-1}$
— The randomized hash value, which is an integer r such that $0 \le r < n$

Assumptions: Public key (n, u, v, k) is valid; $0 \le f < 2^{k-1}$; $0 \le r < n$.

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

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

1) Let $g = \exp(u, f) \times \exp(v, r) \bmod n$.
2) Output g as the encrypted message representative.

Conformance region recommendation: A conformance region should include:

— At least one valid OU public key (n, u, v, k)
— All message representatives f in the range $[0, 2^{k-1} - 1]$

NOTE—The randomized hash value r should be different for encryptions of different message representatives f, as its reuse can lead to the recovery of the partial or total information about the message representative f.

8.2.11 IFDP-OU

IFDP-OU is the OU Decryption Primitive. It is based on the work of Okamoto and Uchiyama [OU98]. It is used in the scheme IFES-EPOC as part of decrypting a message encrypted with the use of IFEP-OU, given the encrypted message representative and the private key of the recipient.

Input:

— The recipient's OU private key (p, q, k, w)
— The encrypted message representative, which is an integer g such that $0 \le g < n$, where n is from the corresponding public key

Assumptions: Private key (p, q, k, w) is valid; $0 \le g < n$.

Output: The message representative, which is an integer f such that $0 \le f < 2^{k-1}$; or "invalid"

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

1) Compute $g_p = \exp(g, p-1) \bmod p^2$.
2) Compute $L(g_p) = (g_p - 1)/p$.
3) Compute $f = L(g_p)/w \bmod p$.
4) Check whether $0 \le f < 2^{k-1}$.
5) If it holds, output f. Otherwise, output "invalid."

Conformance region recommendation: A conformance region should include:

— At least one valid OU private key (p, q, k, w)
— All encrypted message representatives g in the range $[0, n-1]$

8.2.12 IFSP-ESIGN

IFSP-ESIGN is ESIGN Signature Primitive. It is based on the work of Fujisaki and Okamoto [FO98] and Okamoto [Oka90]. It can be invoked in a scheme to compute a signature on a message representative with the private key of the signer, in such a way that the message representative can be recovered from the signature using the public key of the signer by the IFVP-ESIGN primitive. It is intended for a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature generation.

Input:

— The signer's ESIGN private key (p, q, k, e)
— The message representative, which is an integer f such that $0 \le f < 2^{k-1}$

Assumptions: Private key (p, q, k, e) is valid; $0 \le f < 2^{k-1}$.

Output: The signature, which is an integer s such that $0 \le s < n$, where n is from the corresponding public key

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

1) Generate a random integer r, $0 \le r < pq$, such that GCD $(r, n) = 1$.
2) Compute an integer $z = f \times 2^{2k}$.
3) Compute an integer $a = (z - \exp(r, e)) \bmod n$.
4) Compute an integer $w_0 = \lceil a/pq \rceil$.
5) Compute an integer $w_1 = w_0 \times pq - a$.
6) If $w_1 \ge 2^{2k-1}$, then go to step 1).
7) Compute an integer $t = w_0 / (e \times \exp(r, e-1)) \bmod p$.
8) Compute an integer $s = r + tpq \bmod n$.
9) Output s as the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid ESIGN private key (p, q, k, e)
— All message representatives f in the range $[0, 2^{k-1} - 1]$

NOTES

1—The integer r in step 1) should be a one-time value that is generated by the signer following the security recommendations of D.6 and D.7. A new integer r should be generated for every signature, as its reuse can lead to the recovery of the private key. The integer r and other intermediate values should be securely deleted after they are used as their recovery by an opponent may lead to the recovery of the private key.

2—The integer r in step 1), the value $\exp(r, e) \bmod n$ in step 3), and the value $1/(e \times \exp(r, e-1)) \bmod p$ in step 7) may be precomputed in advance of the availability of the message representative, although such precomputation will only be useful for a fraction of the message representatives if the optional test in step 6) is included.

3—Note that the length in bits of the modulus n must be a multiple of 3 (as is defined in 8.1.3.4) in order for the signature and verification primitives to operate correctly. Otherwise, it is possible that some signatures generated by IFSP-ESIGN may be rejected by IFVP-ESIGN, as noted by Shipsey [Shi01].

8.2.13 IFVP-ESIGN

IFVP-ESIGN is ESIGN Verification Primitive. It is based on the work of Fujisaki and Okamoto [FO98] and Okamoto [Oka90]. This primitive recovers the message representative that was signed with IFSP-ESIGN, given only the signature and public key of the signer. It is intended for a signature scheme with appendix and can be invoked in the scheme IFSSA as part of signature verification.

Input:

— The signer's ESIGN public key (n, e, k)
— The signature to be verified, which is an integer s

Assumptions: Public key (n, e, k) is valid.

Output: The message representative, which is an integer f such that $\leq f < 2^{k-1}$; or "invalid"

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

1) If s is not in $[0, n-1]$, output "invalid" and stop.
2) Let $f = \lfloor (\exp(s, e) \bmod n) / 2^{2k} \rfloor$.
3) If f is not in $[0, 2^{k-1} - 1]$, output "invalid" and stop.
4) Output f.

Conformance region recommendation: A conformance region should include:

— At least one valid ESIGN public key (n, e, k)
— All purported signatures s that can be input to the implementation; this should at least include all s in the range $[0, n-1]$

9. Key agreement schemes

9.2 DL/ECKAS-DH1

Insert the following sentence after the first paragraph:

Figure 2 illustrates the scheme.

Insert the following figure:

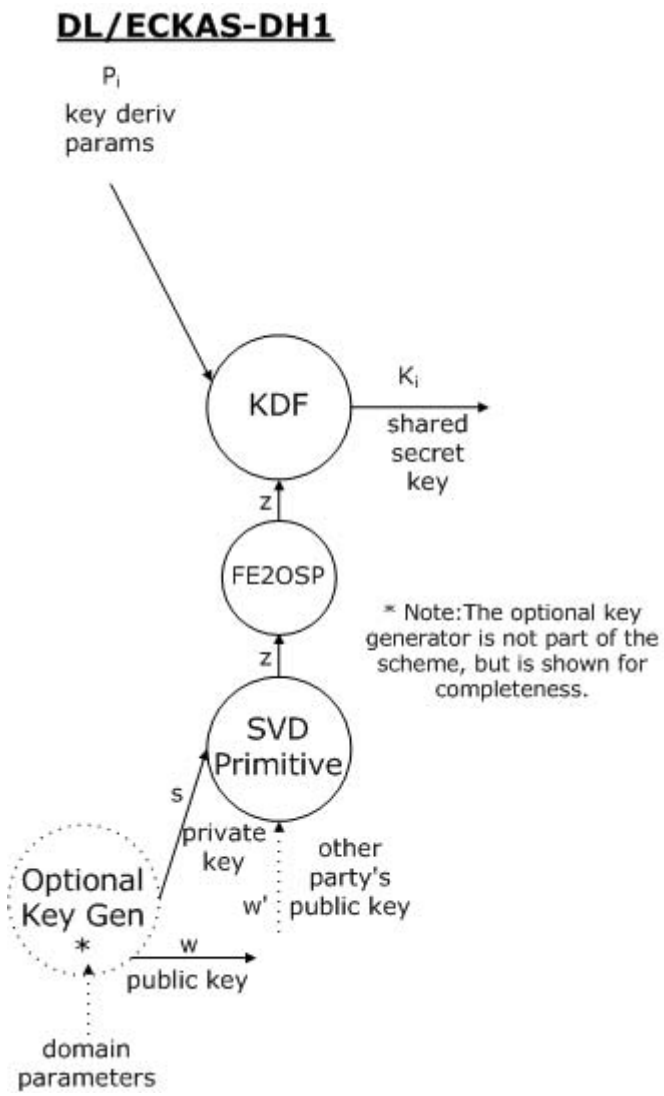

Figure 2—DL/ECKAS-DH1 key agreement operation

9.2.1 Scheme options

Replace the list of options with the following text:

a) A secret value derivation primitive, which shall be DLSVDP-DH, DLSVDP-DHC, ECSVDP-DH, or ECSVDP-DHC
b) For a -DHC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -DH primitive is desired
c) A key derivation function, which should be KDF1 (see 13.1), KDF2 (see 13.2), or a function designated for use with DL/ECKAS-DH1 in an amendment to this standard

9.3 DL/ECKAS-DH2

Insert the following sentence after the first paragraph:

Figure 3 illustrates the scheme.

Insert the following figure:

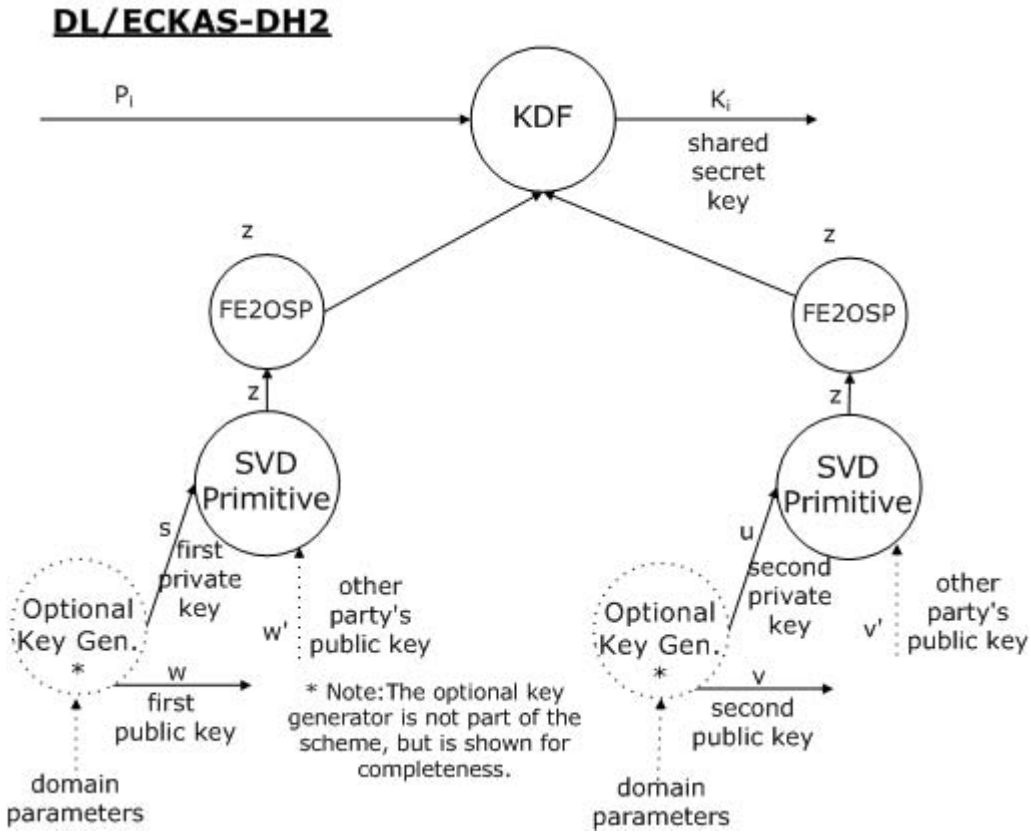

Figure 3—DL/ECKAS-DH2 key agreement operation

9.3.1 Scheme options

Replace the list of options with the following text:

a) Two secret value derivation primitives, each of which (independently) shall be DLSVDP-DH, DLSVDP-DHC, ECSVDP-DH, or ECSVDP-DHC
b) For each -DHC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -DH primitive is desired
c) If the two secret value derivation primitives are the same (and have the same compatibility option in the -DHC case), an indication whether compatibility with DL/ECKAS-DH1 is desired
d) A key derivation function, which should be KDF1 (see 13.1), KDF2 (see 13.2), or a function designated for use with DL/ECKAS-DH2 in an amendment to this standard

9.4 DL/ECKAS-MQV

Insert the following sentence after the first paragraph:

Figure 4 illustrates the scheme.

Insert the following figure:

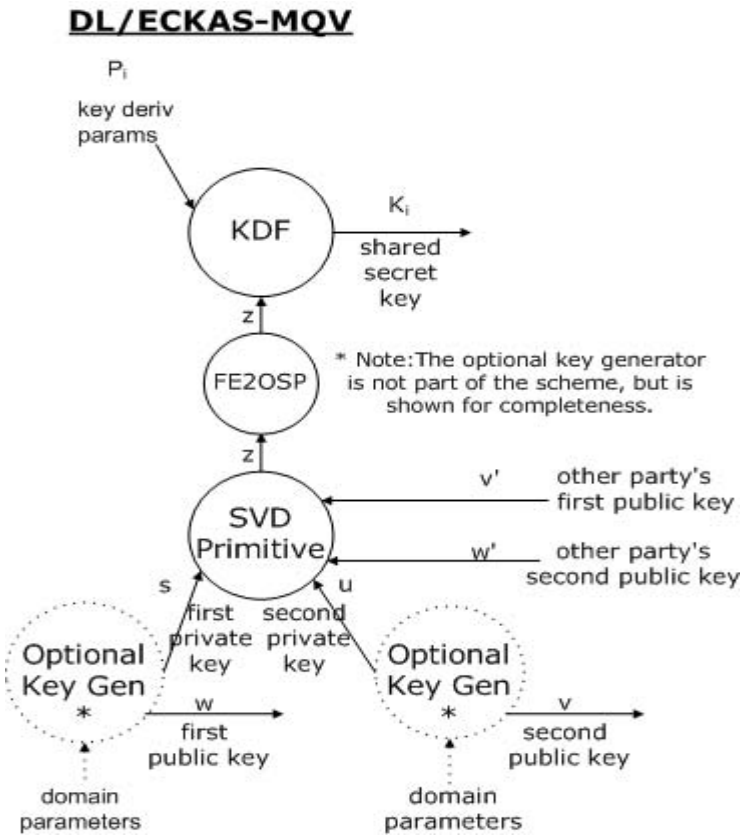

Figure 4—DL/ECKAS-DH2 key agreement operation

9.4.1 Scheme options

Replace the list of options with the following text:

a) A secret value derivation primitive, which shall be DLSVDP-MQV, DLSVDP-MQVC, ECSVDP-MQV, or ECSVDP-MQVC

b) For an -MQVC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -MQV primitive is desired

c) A key derivation function, which should be KDF1 (see 13.1), KDF2 (see 13.2), or a function designated for use with DL/ECKAS-MQV in an amendment to this standard

9.4.2 Key agreement options

Change step 5) to the following:

5) Compute a shared secret value *z* from the selected private key~~s~~ *s*, ~~and u~~ the selected key pair (*u*, *v*), and the other party's two public keys *w*' and *v*' with the selected secret value derivation primitive (see 9.4.1).

10. Signature schemes

Replace the first paragraph with the following text:

The general model for signature schemes is given in 10.1. Five specific schemes and their allowable options are given in 10.2, 10.3, 10.4, 10.5, and 10.6.

10.1 General model

Replace the second paragraph with the following text:

There are two types of signature scheme. In a *signature scheme with appendix*, the signer conveys both the message and the signature to the verifier, who verifies their consistency. In a *signature scheme giving message recovery*, a message is processed in two parts: a *recoverable message part* that can be recovered from the signature and a *nonrecoverable message part.* The signer conveys the signature and the nonrecoverable message part to the verifier, who verifies their consistency and recovers the recoverable part. Such a scheme is said to provide *total message recovery* if the nonrecoverable part is empty and *partial message recovery* otherwise.

A signature scheme giving message recovery will potentially be more efficient in terms of the message expansion for signing a message than a signature scheme with appendix. A signature scheme giving message recovery is typically considered for shorter messages, for which the message expansion is relatively a larger overhead. In a signature scheme with appendix, all of the message will be handled the same way, which may be more convenient in terms of implementation and integration with other operations, such as multiple signatures on the same message.

Replace the third and fourth paragraphs (including their lists) with the following text:

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

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

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

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

A signature generation operation in a signature scheme giving message recovery has the following form:

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

A signature verification operation in a signature scheme giving message recovery has the following form:

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

Replace Note 3 with the following text:

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

— Private key not found in signer's step 1)
— Message, recoverable message part, non-recoverable message part, or private key not supported in signer's step 2)
— Public key not found in verifier's step 1)
— Public key not valid in verifier's step 2)
— Message, non-recoverable message part, signature, or public key not supported in verifier's step 3)

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

10.2 DL/ECSSA

Insert the following sentence after the first paragraph:

Figure 5 illustrates the scheme.

Insert the following figure:

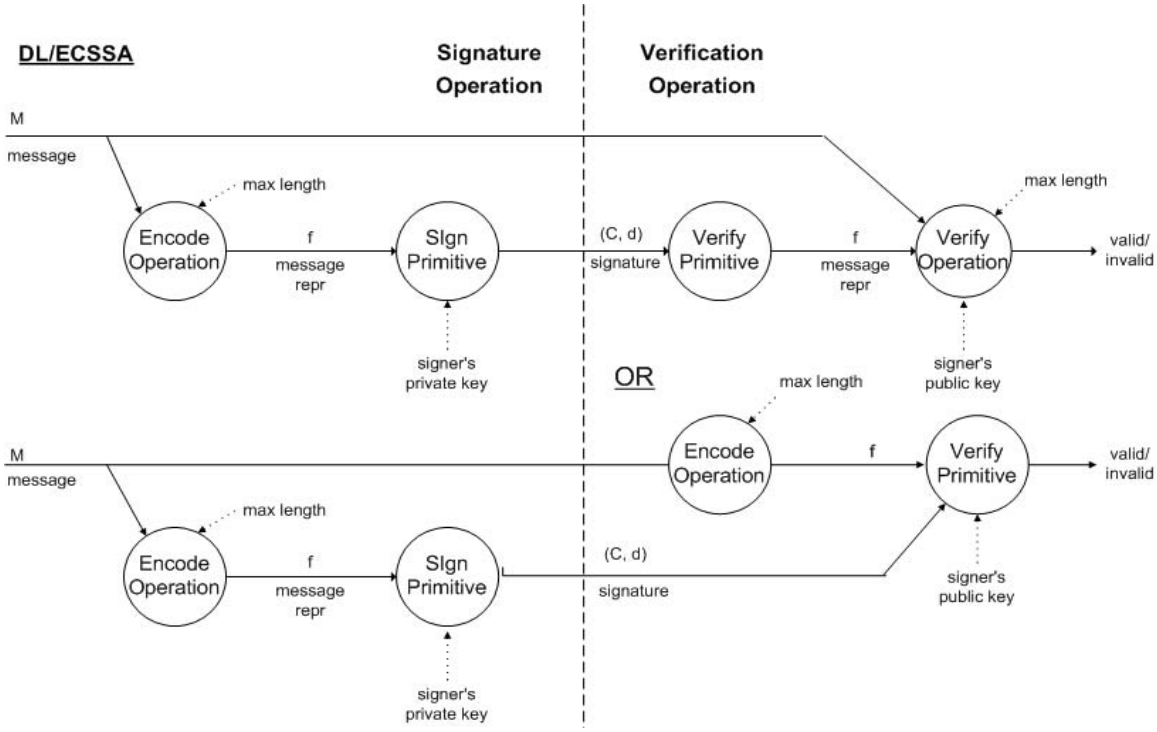

Figure 5—DL/ECSSA signature generation and verification operations

10.2.3 Signature verification operation

Change step 4b) to the following:

b Use the ~~decoding~~ encoding operation of the selected message-encoding method (see 10.2.1) to produce a message representative *f* of maximum length *l* from the message *M* (*f* will be a non-negative integer).

10.3 IFSSA

Insert the following sentence after the first paragraph:

Figure 6 illustrates the scheme.

Insert the following figure:

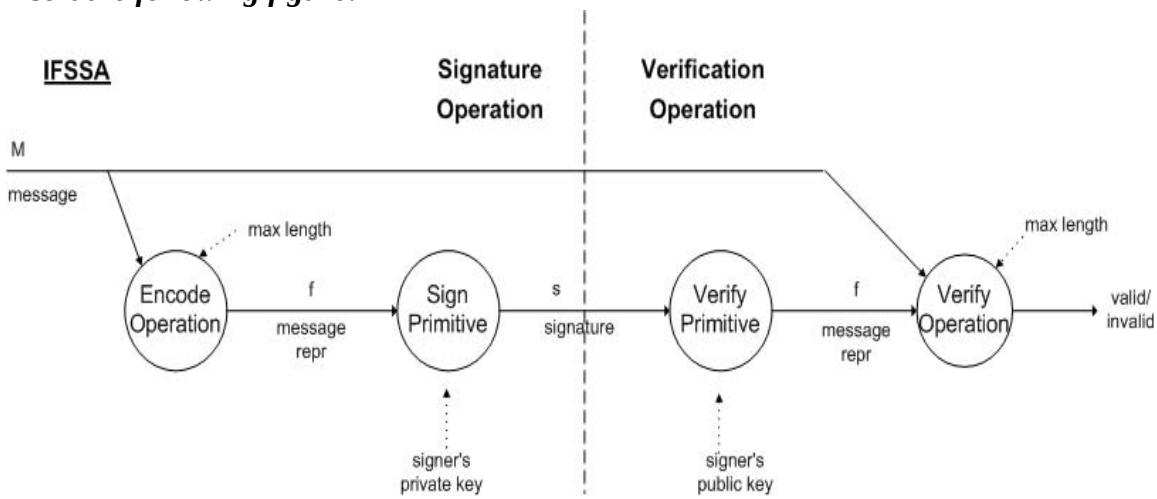

Figure 6—IFSSA signature generation and verification operations

10.3.1 Scheme options

Replace the list of options with the following text:

a) The type of key pair for the signer (RSA, RW, or ESIGN)
b) The signature and verification primitives, which shall be a pair from the following list:
   1) If the signer's public key is an RSA public key, then the primitives shall be either the pair IFSP-RSA1 and IFVP-RSA1 or the pair IFSP-RSA2 and IFVP-RSA2
   2) If the signer's public key is an RW public key, then the primitives shall be the pair IFSP-RW and IFVP-RW.
   3) If the signer's public key is an ESIGN public key, then the primitives shall be the pair IFSP-ESIGN and IFVP-ESIGN.
c) The message-encoding method for signatures with appendix, which should be one of the following:
   1) If the signature primitive is IFSP-RSA1, then the message-encoding method should be EMSA2 (see 12.1.2), EMSA3 (see 12.1.3), EMSA4 (see 12.1.4), or a technique designated for use with IFSSA and this signature primitive in an amendment to this standard.
   2) If the signature primitive is IFSP-RSA2 or IFSP-RW, then the message-encoding method should be EMSA2, EMSA4, or a technique designated for use with IFSSA and this signature primitive in an amendment to this standard (where the message-encoding method shall always produce a message representative congruent to 12 modulo 16).
   3) If the signature primitive is IFSP-ESIGN, then the message-encoding method should be EMSA5 or a technique designated for use with IFSSA and this signature primitive in an amendment to this standard.

NOTE—EMSA2 requires that the message representative length, i.e., (length of *n* in bits) – 1 for RSA or RW, be congruent to 7 mod 8.

Replace 10.3.2 with the following text:

10.3.2 Signature generation option

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

1) Select a valid private key *K* for the operation.
2) If the selected signature primitive is IFSP-RSA1, IFSP-RSA2, or IFSP-RW, then set the maximum length of the message representative to be *l* = (length of *n* in bits) – 1, where *n* is the modulus in the RSA or RW private key. If the selected signature primitive is IFSP-ESIGN, then set the maximum length to be *l* = *k* – 1, where *k* is from the ESIGN private key.
3) Use the encoding operation of the selected message-encoding method (see 10.3.1) to produce a message representative *f* of maximum length *l* from the message *M* (*f* will be a non-negative integer). If the selected encoding method is EMSA2 or EMSA4, *f* will be congruent to 12 modulo 16.
4) Apply the selected signature primitive (see 10.3.1) to the integer *f* and the selected private key *K* to generate the signature *s*.
5) Output the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid private key *K*
— A range of messages *M*

Replace 10.3.3 with the following text:

10.3.3 Signature verification option

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

1) Obtain the signer's purported public key.
2) *(Optional)* Validate the public key. Output "invalid" and stop if validation fails.
3) Apply the selected verification primitive (see 10.3.1) to the signature *s* and the signer's public key to recover an integer *f*. If the output of the primitive is "invalid," output "invalid" and stop.
4) If the selected verification primitive is IFVP-RSA1, IFVP-RSA2, or IFVP-RW, then set the maximum length of the message representative to be *l* = (length of *n* in bits) – 1, where *n* is the modulus in the RSA or RW public key. If the selected verification primitive is IFVP-ESIGN, then set the maximum length to be *l* = *k* – 1, where *k* is from the ESIGN public key.
5) Verify that the integer *f* is a correct encoded representative of the message *M* using the verification operation of the selected message-encoding method (see 10.3.1) and maximum length *l*. If that is the case, output "valid." Else, output "invalid."

Conformance region recommendation: A conformance region should include:

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

NOTES

1—The upper bound on the length in bits of the message representative is so that the message representative is less than the modulus *n*, as required by the RSA and RW signature primitives, or so that it is less than 2^{k-1}, as required by the ESIGN signature primitive. As message-encoding methods for signatures with appendix are usually based on hash functions, the length of a message that can be signed by this scheme is usually unrestricted or constrained by a very large number.

2—This scheme, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.31-1998 [B7], ISO/IEC 14888-3:1998 [B80], and PKCS #1 v2.1 [PKCS1v2_1].

Insert the following three subclauses (10.4–10.6) after 10.3:

10.4 DL/ECSSR

DL/ECSSR is Discrete Logarithm and Elliptic Curve Signature Scheme with Recovery.

Figure 7 illustrates the scheme.

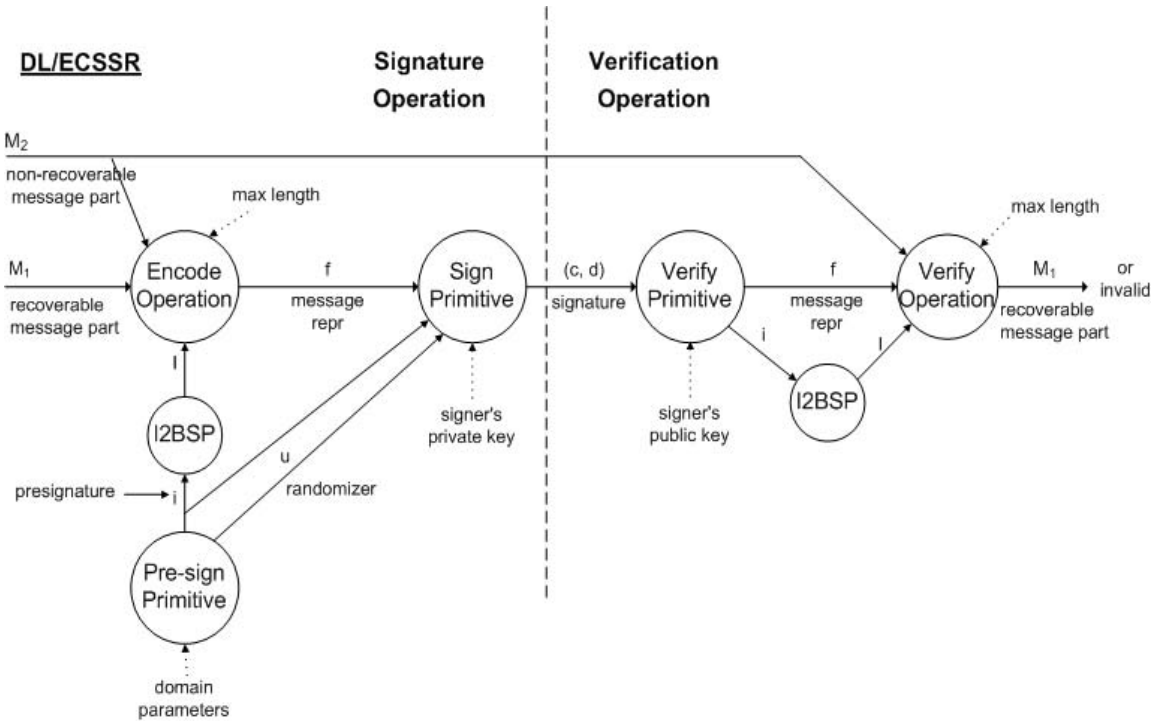

Figure 7—DL/ECSSR signature generation and verification operations

10.4.1 Scheme options

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

a) The pre-signature, signature and verification primitives, which shall be one of the following triples of primitives:
   1) DLPSP-NR2/PV, DLSP-NR2, and DLVP-NR2
   2) ECPSP-NR2/PV, ECSP-NR2, and ECVP-NR2

b) The message-encoding method for signatures giving message recovery, which should be EMSR1 (see 12.3.1), or a technique designated for use with DL/ECSSR in an amendment to this standard

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

10.4.2 Signature generation option

A signature (c, d) shall be generated by a signer from a recoverable message part M_1 and a (possibly empty) nonrecoverable message part M_2 by the following or an equivalent sequence of steps:

1) Select a valid private key s and its associated set of domain parameters for the operation.
2) Apply the selected pre-signature primitive (see 10.4.1) to generate a randomizer u and a pre-signature i. Convert the pre-signature i to a bit string I of length $\lceil \log_2 q \rceil$ bits using I2BSP.
3) Set the maximum length of the message representative to be l = (length of r in bits) – 1, where r is the order of the base point in the DL or EC set of domain parameters.
4) Use the encoding operation of the selected message-encoding method (see 10.4.1) to produce a message representative f of maximum length l (f will be a non-negative integer) from the recoverable message part M_1, the nonrecoverable message part M_2, and the bit string I.
5) Apply the selected signature primitive (see 10.4.1) to the integer f, the private key s, the randomizer u, and the pre-signature i to generate the signature (c, d). If the signature primitive outputs "error," then go to step 2).
6) Output the signature and optionally the length of the recoverable message part M_1.

Conformance region recommendation: A conformance region should include:

— At least one valid set of domain parameters
— At least one valid private key s for each set of domain parameters
— A range of message parts M_1 and M_2 (where the range of nonrecoverable message parts may include only the empty string; i.e., conformance may be claimed for total message recovery only, and where the range of recoverable message parts may explicitly *exclude* the empty string, i.e., to distinguish from a signature scheme with appendix)

NOTE—If the one-time key pair in step 2) is generated deterministically as a function of the message and the private key as suggested in D.5.2.1, and in step 5) the selected signature primitive outputs "error," the signature generation operation above will enter an infinite loop. As an "error" output is an extremely rare event for recommended sizes of domain parameters, this is not a practical concern, and it only applies in the case that the one-time key pair is generated deterministically, not when it is generated at random. If desired, the concern can be addressed by including a counter in the key derivation parameters for generating the one-time key pair deterministically, as noted in D.5.2.1.

10.4.3 Signature verification option

A signature (c, d) shall be verified and the recoverable message part M_1 recovered by a verifier, given a (possibly empty) nonrecoverable message part M_2 and optionally the length of the recoverable message part, by the following or an equivalent sequence of steps:

1) Obtain the signer's purported public key w and its associated set of domain parameters for the operation.
2) *(Optional)* Validate the public key w and its associated set of domain parameters. Output "invalid" and stop if validation fails.
3) Apply the selected verification primitive (see 10.4.1) to the signature (c, d) and the signer's public key to recover a message representative f and a pre-signature i (both will be non-negative integers). If the output of the primitive is "invalid," output "invalid" and stop. Convert the pre-signature i to a bit string I of length $\lceil \log_2 q \rceil$ bits using I2BSP.
4) Set the maximum length of the message representative to be l = (length of r in bits) – 1, where r is the order of the base point in the DL or EC set of domain parameters.
5) Use the decoding operation of the selected message-encoding method (see 10.4.1) to verify that the integer f is a correct encoded representative according to the encoding method and maximum length l, and to recover the recoverable message part M_1 given the (possibly empty) nonrecoverable message part M_2, optionally the length of the recoverable message part, and the bit string I. If the output of the decoding operation is "invalid," output "invalid." Otherwise, output the recoverable message part M_1.

Conformance region recommendation: A conformance region should include:

— At least one valid set of domain parameters
— At least one valid public key w for each set of domain parameters; if key validation is performed, invalid public keys w that are appropriately rejected by the implementation may also be included in the conformance region
— All nonrecoverable message parts M_2 that can be input to the implementation (which may include only the empty string, i.e., conformance may be claimed for total message recovery only)
— All purported signatures (c, d) that can be input to the implementation and that correspond to some range of recoverable message parts (for instance, where the range of recoverable message parts may exclude the empty string); this should at least include all (c, d) that correspond to the range of recoverable messages such that c and d are in the range [0, r – 1], where r is from the domain parameters of w

NOTES

1—As the recommended message-encoding method is based on a hash function, the length of the nonrecoverable message part that can be signed by this scheme is either unrestricted or constrained by a very large number. However, the length of the recoverable message part is limited.

2—These schemes, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ISO/IEC 9796-3:2000 [B79].

3—The length of the recoverable message part must be known in advance to the verifier in this scheme if the encoding method is EMSR1.

10.5 DL/ECSSR-PV

DL/ECSSR-PV is Discrete Logarithm and Elliptic Curve Signature Scheme with Recovery, Pintsov–Vanstone version.

Figure 8 illustrates the scheme.

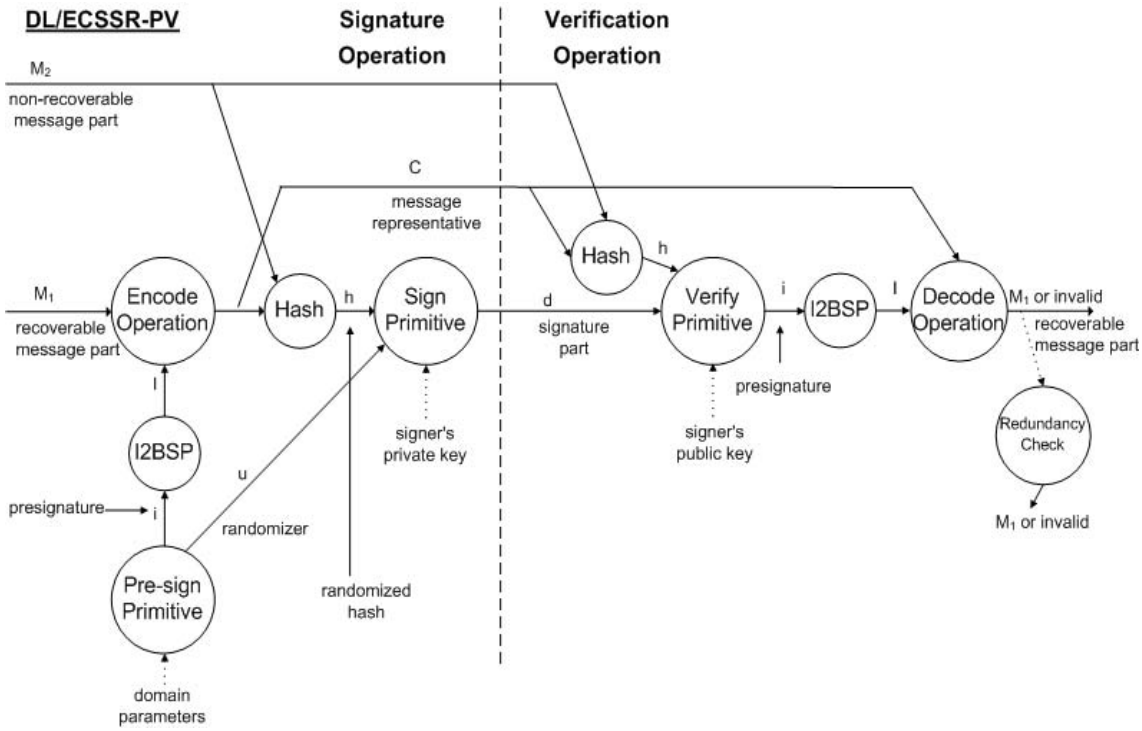

Figure 8—DL/ECSSR-PV signature generation and verification operations

10.5.1 Scheme options

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

a) The pre-signature, signature and verification primitives, which shall be one of the following triples of primitives:
   1) DLPSP-NR2/PV, DLSP-PV, and DLVP-PV
   2) ECPSP-NR2/PV, ECSP-PV, and ECVP-PV
b) The message-encoding method for signatures giving message recovery, which should be EMSR2 (see 12.3.2) (including the encoding parameter *padLen*, which is an integer between 1 and 255 giving the amount of added redundancy in octets, and including other parameters of EMSR2), or a technique designated for use with DL/ECSSR-PV in an amendment to this standard
c) The redundancy criteria necessary for acceptance of the message after it has been recovered and successfully decoded (see D.5.2.2)
d) The hash function *Hash*, which shall be one of the hash functions in 14.1 or a technique designated for use with DL/ECSSR-PV in an amendment to this standard

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

10.5.2 Signature generation option

A signature (C, d) shall be generated by a signer from a recoverable message part M_1 and a (possibly empty) nonrecoverable message part M_2 by the following or an equivalent sequence of steps:

1) Verify that the message (consisting of the recoverable message part M_1 and the nonrecoverable message part M_2) meets the agreed redundancy criteria. If it does not, output "error."
2) Select a valid private key s and its associated set of domain parameters for the operation.
3) Apply the selected pre-signature primitive (see 10.5.1) to generate a randomizer u and a pre-signature i. Convert the pre-signature i to an octet string I of length $\lceil \log_{256} q \rceil$ octets using I2OSP.
4) Use the encoding operation of the selected message-encoding method (see 10.5.1) to produce a message representative C from the recoverable message part M_1 and the octet string I.
5) Apply the selected hash function (see 10.5.1) to the message representative C and the nonrecoverable message part M_2 to generate a hash value H as $H = Hash\,(C \parallel M_2)$.
6) Convert the hash value H to an integer h with the primitive OS2IP.
7) Apply the selected signature primitive (see 10.5.1) to the hash value h, the private key s, and the randomizer u to generate a signature part d.
8) Output (c, d) as the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid set of domain parameters
— At least one valid private key s for each set of domain parameters
— A range of messages parts M_1 and M_2 (where the range of nonrecoverable message parts may include only the empty string, i.e., conformance may be claimed for total message recovery only, or where the range of recoverable message parts may explicitly *exclude* the empty string, i.e., to distinguish from a signature scheme with appendix)

10.5.3 Signature verification option

A signature (C, d) shall be verified and the recoverable message M_1 recovered by a verifier, given a (possibly empty) nonrecoverable message part M_2, by the following or an equivalent sequence of steps:

1) Obtain the signer's purported public key w' and its associated set of domain parameters for the operation.
2) *(Optional)* Validate the public key w' and its associated set of domain parameters. Output "invalid" and stop if validation fails.
3) Apply the selected hash function (see 10.5.1) to the message representative C and the nonrecoverable message part M_2 to generate a hash value H as $H = Hash\,(C \parallel M_2)$.
4) Convert the hash value H to an integer h with the primitive OS2IP.
5) Apply the selected verification primitive (see 10.5.1) to the signature part d, the integer h, and the signer's public key w to recover a pre-signature i. Convert the pre-signature i to an octet string I of length $\lceil \log_{256} q \rceil$ octets using I2OSP.
6) Use the decoding operation of the selected message-encoding method (see 10.6.1) to verify that the message representative C is a correctly encoded representative according to the encoding method, and to recover the recoverable message part M_1, given the octet string I. If the output of the decoding operation is "invalid," output "invalid."
7) Verify that the message (consisting of the recoverable message part M_1 and the nonrecoverable message part M_2) meets the agreed redundancy criteria. If it does not, output "invalid." Otherwise, output the recoverable message part M_1.

Conformance region recommendation: A conformance region should include:

- At least one valid set of domain parameters
- At least one valid public key w' for each set of domain parameters; if key validation is performed, invalid public keys w' that are appropriately rejected by the implementation may also be included in the conformance region
- All nonrecoverable message parts M_2 that can be input to the implementation (which may include only the empty string, i.e., conformance may be claimed for total message recovery only)
- All purported signatures (C, d) that can be input to the implementation and that correspond to some range of recoverable message parts (for instance, where the range of recoverable message parts may exclude the empty string); this should at least include all (C, d) that correspond to the range of recoverable messages such that d is in the range $[0, r - 1]$, where r is from the domain parameters of w'

NOTES

1—The lengths of both the recoverable message part and the nonrecoverable message part that can be signed by this scheme are unrestricted or constrained by a very large number.

2—If the encoding method is EMSR2, the agreed redundancy criteria should ensure that no recoverable message part that meets the agreed redundancy criteria is a prefix of another recoverable message part that meets the agreed redundancy criteria, or comparable provisions should be made to address the possibility of ambiguity in the encoding $C \| M_2$ in the input to the hash function (see D.5.2.2.2 Note 4).

3—These schemes, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ISO/IEC 15946-4 [ISO-15946-4].

4—The length of the recoverable message part does not need to be known in advance to the verifier in this scheme if the encoding method is EMSR2.

10.6 IFSSR

IFSSR is Integer Factorization Signature Scheme with Recovery.

Figure 9 illustrates the scheme.

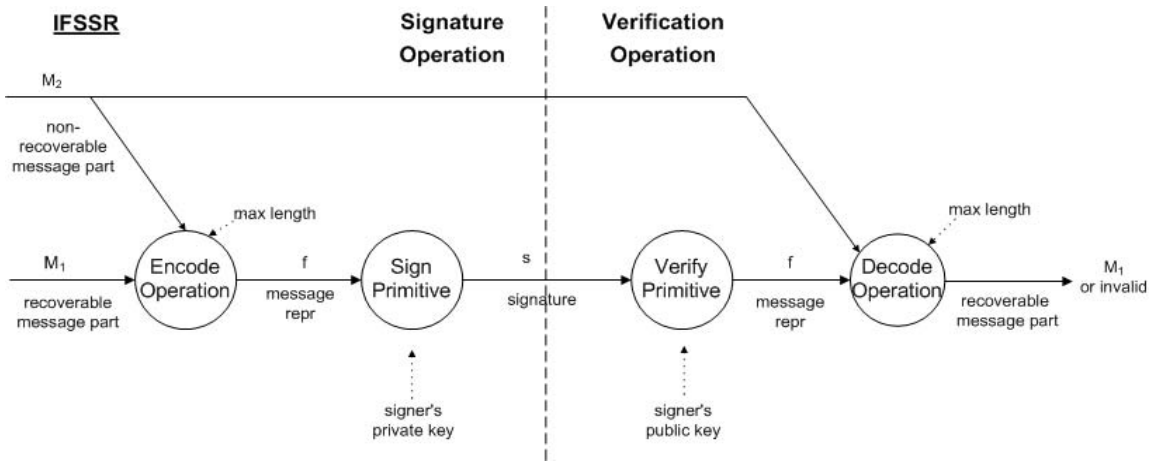

Figure 9—IFSSR signature generation and verification operations

10.6.1 Scheme options

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

a) The type of key pair for the signer (RSA or RW).

b) The signature and verification primitives, which shall be a pair from the following list:

   1) If the signer's public key is an RSA public key, then the primitives shall be either the pair IFSP-RSA1 and IFVP-RSA1 or the pair IFSP-RSA2 and IFVP-RSA2.

   2) If the signer's public key is an RW public key, then the primitives shall be the pair IFSP-RW and IFVP-RW.

c) The message-encoding method for signatures giving message recovery, which should be EMSR3 (see 12.3.4), or a technique designated for use with IFSSR in an amendment to this standard (if the signature primitive is IFSP-RSA2 or IFSP-RW, the message-encoding method shall always produce message representatives congruent to 12 modulo 16).

The information in this subclause may remain the same for any number of executions of the signature scheme, or it may be changed at some frequency. The information can be shared among a number of parties.

NOTE—EMSR3 produces message representatives congruent to 12 modulo 16 and is therefore appropriate for use with IFSP-RSA2 and IFSP-RW as well as IFSP-RSA1. For EMSR3, the recoverable part of the message is treated as a bit string.

10.6.2 Signature generation option

A signature s shall be generated by a signer from a recoverable message part M_1 and a (possibly empty) non-recoverable message part M_2 by the following or an equivalent sequence of steps:

1) Select a valid private key K for the operation.

2) Set the maximum length of the message representative to be l = (length of n in bits) – 1, where n is the modulus in the IF private key.

3) Use the encoding operation of the selected message-encoding method (see 10.6.1) to produce a message representative f of maximum length l (f will be a non-negative integer) from the recoverable message part M_1 and the nonrecoverable message part M_2. If the selected signature primitive is IFSP-RSA2 or IFSP-RW, f will be congruent to 12 modulo 16.

4) Apply the selected signature primitive (see 10.6.1) to the integer f and the selected private key K to generate the signature s.

5) Output the signature.

Conformance region recommendation: A conformance region should include:

— At least one valid private key K

— A range of message parts M_1 and M_2 (where the range of nonrecoverable message parts may include only the empty string, i.e., conformance may be claimed for total message recovery only, and where the range of recoverable message parts may include octet strings only, i.e., arbitrary-length bit strings do not need to be supported, or may explicitly *exclude* the empty string, i.e., to distinguish from a signature scheme with appendix)

10.6.3 Signature verification operation

A signature *s* shall be verified and the recoverable message part M_1 recovered by a verifier, given a (possibly empty) nonrecoverable message part M_2, by the following or an equivalent sequence of steps:

1) Obtain the signer's purported public key (*n*, *e*).
2) *(Optional)* Validate the public key (*n*, *e*). Output "invalid" and stop if validation fails.
3) Apply the selected verification primitive (see 10.6.1) to the signature *s* and the signer's public key to recover an integer *f*. If the output of the primitive is "invalid," output "invalid" and stop.
4) Set the maximum length of the message representative to be *l* = (length of *n* in bits) – 1.
5) Use the decoding operation of the selected message-encoding method (see 10.6.1) to verify that the integer *f* is a correct encoded representative according to the encoding method and maximum length *l*, and to recover the recoverable message part M_1 given the (possibly empty) nonrecoverable message part M_2. If the output of the decoding operation is "invalid," output "invalid." Otherwise, output the recoverable message part M_1.

Conformance region recommendation: A conformance region should include:

— At least one valid public key (*n*, *e*); if key validation is performed, invalid public keys (*n*, *e*) that are appropriately rejected by the implementation may also be included in the conformance region
— All nonrecoverable message parts M_2 that can be input to the implementation (which may include only the empty string, i.e., conformance may be claimed for total message recovery only)
— All purported signatures *s* that can be input to the implementation and that correspond to some range of recoverable message parts (for instance, where the range of recoverable message parts may include octet strings only); this should at least include all *s* in the range [0, *n* – 1] for IFVP-RSA1 or [0, (*n* – 1)/2] for IFVP-RSA2 and IFVP-RW that correspond to the range of recoverable messages

NOTES

1—As the recommended message-encoding method is based on a hash function, the length of the nonrecoverable message part that can be signed by this scheme is either unrestricted or constrained by a very large number. However, the length of the recoverable message part is limited.

2—This scheme, with appropriate restrictions on the scheme options and inputs, is compatible with techniques being considered in the draft revision to ISO 9796-2 [ISO-9796-2-revision].

3—The length of the recoverable message part does not need to be known in advance to the verifier in this scheme.

11. Encryption schemes

11.1 General model

Replace the last paragraph with the following text:

An encryption operation has the following form for all of the schemes:

1) Obtain the recipient's purported public key (together with its set of domain parameters, if any) for the operation (Note 1 below).
2) *(Optional)* Validate the public key and its associated set of domain parameters, if any. If validation fails, output "invalid" and stop (Note 2 below).
3) Apply certain cryptographic operations to the message, the public key, and the encoding parameters, if any, to produce a ciphertext.
4) Output the ciphertext.

A decryption operation has the following form:

1) Select a valid private key (together with its set of domain parameters, if any) for the operation.
2) Apply certain cryptographic operations to the ciphertext, the private key, and the encoding parameters, if any, to recover a message (Note 2 below).
3) Output the message or "invalid" according to the result of step 2).

NOTES

1—*(Authentication of ownership)* The process of obtaining the recipient's public key [step 1) of encryption] may involve authentication of ownership of the public key, as further described in D.3.2 and D.5.3.4. This may be achieved by verifying a certificate or by other means. The means by which the key (or the certificate containing it) is obtained may vary, and they may include the recipient sending the key to the sender, or the sender obtaining the key from a third party or from a local database.

2—*(Domain parameter and key validation)* As the underlying primitives generally assume that the domain parameters (if any) and public key are valid, the result of a primitive may be undefined otherwise. Consequently, it is recommended that the sender validate the public key (and its associated set of domain parameters, if any) in step 1) of the encryption operation, unless the risk of operating on invalid domain parameters or keys is mitigated by other means, as discussed further in D.3.3 and D.5.3.5. Key validation may also be a consideration in step 2) of the decryption operation for some schemes. Examples of "other means" include validation within the supporting key management.

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

— Public key not found in sender's step 1)
— Public key not valid in sender's step 2)
— Message, encoding parameters, or public key not supported in sender's step 3)
— Private key not found in recipient's step 1)
— Ciphertext, encoding parameters, or private key not supported in recipient's step 2)

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

11.2 IFES

Insert the following sentence after the first paragraph:

Figure 10 illustrates the scheme.

Insert the following figure:

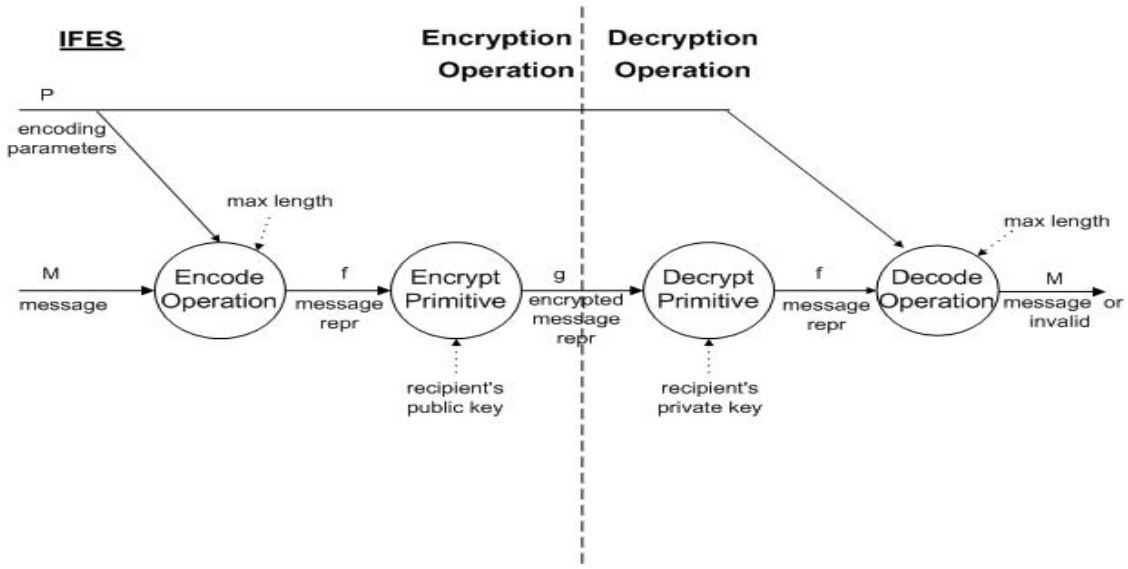

Figure 10—IFES encryption and decryption operations

11.2.2 Encryption operation

Delete the references to Notes 1 and 2 in step 1) and step 2).

11.2.3 Decryption operation

Delete Notes 1–3 and renumber the remaining notes as Notes 1 and 2.

Change the (renumbered) Note 2 to the following:

2—*(Compatibility)* This scheme, with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.44 [B9] and PKCS #1 v2.1 [PKCS1v2_1].

Insert the following two subclauses (11.3 and 11.4) after 11.2:

11.3 DL/ECIES

DL/ECIES is Discrete Logarithm and Elliptic Curve Integrated Encryption Scheme. It is based on the work of Abdalla et al. [ABR98] and ANSI X9.63 [B12].

Figure 11 illustrates the scheme.

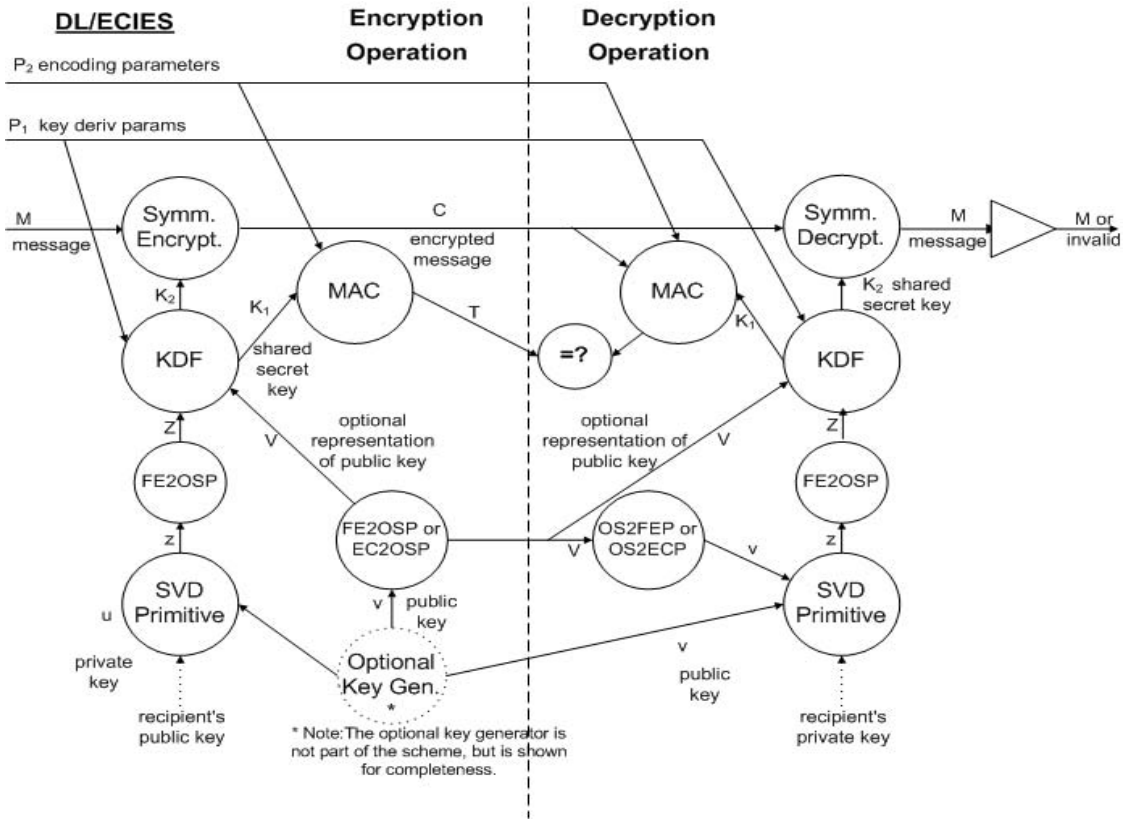

Figure 11—DL/ECIES encryption and decryption operations

11.3.1 Scheme options

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

a) The secret value derivation primitive, which shall be DLSVDP-DH, DLSVDP-DHC, ECSVDP-DH, or ECSVDP-DHC
b) For the -DHC secret value derivation primitive, an indication as to whether or not compatibility with the corresponding -DH primitive is desired
c) The method for encrypting the message, which shall be either:
    1) A stream cipher based on a key derivation function, where the key derivation function should be KDF2 (see 13.2) or a function designated for use with DL/ECIES in an amendment to this standard (this method is only recommended for relatively short messages such as symmetric keys, and in non-DHAES mode, the messages should have a fixed length for a given public key—see D.5.3 Note 1 and D.5.3.2.2)
    2) A key derivation function combined with a symmetric encryption scheme, where the key derivation function should be KDF2 (see 13.2) or a technique designated for use with DL/ECIES in an amendment to this standard, and the symmetric encryption scheme should be one of the schemes in 14.3, or a technique designated for use with DL/ECIES in an amendment to this standard
d) The message authentication code, which should be MAC1 (see 14.4.1), or a MAC designated for use with DL/ECIES in an amendment to this standard

e) An indication as to whether to operate in "DHAES" mode (see Abdalla et al. [ABR98]), i.e., whether to include a representation of the sender's public key as an input to the key derivation function
f) In the EC case, a pair of primitives for converting elliptic curve points to and from octet strings appropriate for the underlying finite field, which should be among those in 5.5.6.2 and 5.5.6.3

The information in this subclause may remain the same for any number of executions of the encryption scheme, or it may be changed at some frequency. The information need not be kept secret. However, it is recommended that for any particular recipient's public key, the same set of options be used each time to avoid unintended interactions between different options for that public key. (See D.5.3 for further discussion.)

NOTES

1—Although defined in terms of bit strings, the recommended symmetric encryption schemes in 14.3 all require that the length of the bit string is divisible by 8. To encrypt a bit string of length not divisible by 8, a different symmetric encryption scheme should be chosen, or the stream cipher option should be selected.

2—Parameters to the underlying symmetric encryption scheme, if any (such as a variable initialization vector), should either be included in the encrypted message *C*, or in the encoding parameters, to ensure their integrity.

11.3.2 Encryption operation

A ciphertext (*V, C, T*) shall be generated by a sender from a message *M* of bit length *l*, key derivation parameters P_1, and encoding parameters P_2 by the following or an equivalent sequence of steps:

1) Establish the valid set of DL or EC domain parameters with which the parties' key pairs shall be associated.
2) Select a key pair (*u*, *v*) for the operation, associated with the parameters established in step 1).
3) Obtain the recipient's purported public key *w*' for the operation, associated with the parameters established in step 1).
4) *(Optional)* If the selected secret value derivation primitive is DLSVDP-DHC or ECSVDP-DHC, then validate that *w*' is an element in the appropriate group (i.e., in *GF*(*q*) for DL or on the elliptic curve for EC—see 6.2.2 and 7.2.2); otherwise validate that *w*' is a valid public key. If any validation fails, output "invalid public key" and stop.
5) Compute a shared secret value *z* from the private key *u* and the recipient's public key *w*' with the selected secret value derivation primitive (see 9.2.1).
6) Convert the shared secret value *z* to an octet string *Z* using FE2OSP.
7) Convert the sender's public key to an octet string *V* using FE2OSP (in the DL case) or the selected primitive for converting an elliptic curve point to an octet string (in the EC case).
8) In DHAES mode, let $VZ = V \| Z$. Otherwise, let $VZ = Z$.
9) If the method for encrypting the message is a stream cipher based on a key derivation function:
   a) Derive a shared secret key *K* from the octet string *VZ* and the key derivation parameters P_1 with the selected key derivation function (see 9.2.1). The length in bits of the shared secret key *K* shall be $l + k_2$ where k_2 is the length in bits of the key for the message authentication code. In non-DHAES mode, let K_1 be the leftmost *l* bits of *K* and let K_2 be the remaining k_2 bits. In DHAES mode, let K_2 be the leftmost k_2 bits of *K* and let K_1 be the remaining *l* bits. (Note that the order of K_1 and K_2 is different in the two modes—see D.5.3 Note 1.)
   b) Let $C = M \oplus K_1$.
10) If the method for encrypting the message is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a shared secret key *K* from the octet string *VZ* and the key derivation parameters P_1 with the selected key derivation function (see 9.2.1). The length in bits of the shared secret key *K* shall be $k_1 + k_2$ where k_1 is the length in bits of the key for the symmetric encryption

scheme and k_2 is the length in bits of the key for the message authentication code. Let K_1 be the leftmost K_1 bits of K, and let K_2 be the remaining k_2 bits.

b) Encrypt M under the key K_1 with the symmetric encryption scheme to produce an encrypted message C.

11) In DHAES mode, convert the length of P_2 in bits to an octet string L_2 of length 8 octets using I2OSP. Otherwise, let L_2 be the empty string.

12) Apply the selected message authentication code to the encrypted message C, the encoding parameters P_2, and the (possibly empty) length L_2 under the key K_2 to produce an authentication tag $T = MAC_{K_2}(C \| P_2 \| L_2)$.

13) Output the triple (V, C, T) as the ciphertext.

Conformance region recommendation: A conformance region should include:

— At least one valid set of domain parameters
— All valid public keys *w*' associated with the set of domain parameters; if key validation is performed or a -DHC primitive is used, invalid public keys that are appropriately handled by the implementation may also be included in the conformance region
— A range of messages M, key derivation parameters P_1, and encoding parameters P_2 (where the ranges of key derivation parameters and encoding parameters may each include only the empty string)

NOTES

1—The key pair in step 2) should generally be a one-time key pair that is generated and stored by the sender following the security recommendations of D.3.1, D.4.1.2, D.6, and D.7. However, for this scheme, the same key pair may also be used in additional encryption operations with the same domain parameters, either for the same or for different recipients, following the ephemeral-static mode of Diffie–Hellman (see Housley [Hou02], Sec. 6.2.2, and Rescorla [Res99], Sec. 2.3). In such a case, the private key should be securely deleted after all of the encryption operations are complete, as its recovery by an opponent can lead to the recovery of messages encrypted using that key pair. If the key pair is used for additional encryption operations for the same recipient, the key derivation parameters P_1 should vary among the operations so that the key K varies (see D.5.3.3).

2—Non-DHAES mode is included for compatibility with ANSI X9.63 [B12] and other standards efforts. In non-DHAES mode, a representation of the sender's public key v is not explicitly included as an input to the key derivation function. As further discussed in D.5.3, this raises mainly theoretical concerns. DHAES mode is offered for increased assurance.

3—In non-DHAES mode, if the method for encrypting the message is a key derivation function combined with a symmetric encryption scheme, provision should be made to address the possibility of ambiguity in the encoding $C \| P_2$ in the input to the MAC in step 12). For example, P_2 can have a fixed length or end with a fixed-length encoding of its length (see D.5.3.3 for further discussion). This ambiguity is avoided in the stream cipher option if the message (and hence C) has a fixed length for a given public key as previously recommended.

In DHAES mode, the ambiguity is avoided by including the length of P_2 in the input. (Note that this protection is not part of DHAES per se as defined by Abdalla et al. [ABR98], because their proposal does not use encoding parameters, but the protection is included for increased assurance.)

11.3.3 Decryption operation

The message M shall be recovered from the ciphertext (V, C, T), key derivation parameters P_1, and encoding parameters P_2 by the following or an equivalent sequence of steps:

1) Establish the valid set of DL or EC domain parameters with which the parties' key pairs shall be associated.
2) Select the private key *s*' for the operation, associated with the parameters established in step 1).

3) Convert the octet string *V* to a public key *v* using OS2FEP (in the DL case) or the selected primitive for converting an octet string to an elliptic curve point (in the EC case). If conversion fails, output "invalid" and stop.
4) *(Optional)* If the selected secret value derivation primitive is DLSVDP-DHC or ECSVDP-DHC, then validate that *v* is an element in the appropriate group (i.e., in $GF(q)$ for DL or on the elliptic curve for EC—see 6.2.2 and 7.2.2); otherwise validate that *v* is a valid public key. If any validation fails, output "invalid public key" and stop.
5) Compute a shared secret value *z* from the private key *s*' and the public key *v* with the selected secret value derivation primitive (see 9.2.1).
6) Convert the shared secret value *z* to an octet string *Z* using FE2OSP.
7) In DHAES mode, let $VZ = V \| Z$. Otherwise, let $VZ = V$.
8) If the method for encrypting the message is a stream cipher based on a key derivation function:
   a) Derive a shared secret key *K* from the octet string *VZ* and the key derivation parameters P_1 with the selected key derivation function (see 9.2.1). The length in bits of the shared secret key *K* shall be $l + k_2$ as above. In non-DHAES mode, let K_1 be the leftmost *l* bits of *K* and let K_2 be the remaining k_2 bits. In DHAES mode, let K_2 be the leftmost k_2 bits of *K* and let K_1 be the remaining *l* bits. (Note that the order of K_1 and K_2 is different in the two modes—see D.5.3 Note 1.)
   b) Let $M = C \oplus K_1$.
9) If the method for encrypting the message is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a shared secret key *K* from the octet string *VZ* and the key derivation parameters P_1 with the selected key derivation function (see 9.2.1). The length in bits of the shared secret key *K* shall be $k_1 + k_2$ as above. Let K_1 be the leftmost k_1 bits of *K*, and let K_2 be the remaining k_2 bits.
   b) Decrypt *C* under the key K_1 with the symmetric encryption scheme to recover *M*. (If the decryption operation outputs "error," output "invalid" and stop.)
10) In DHAES mode, convert the length of P_2 in bits to an octet string L_2 of length 8 octets using I2OSP. Otherwise, let L_2 be the empty string.
11) Apply the selected message authentication code to the encrypted message *C*, the encoding parameters P_2 and the (possibly empty) length P_2 to verify the authentication tag *T*, i.e., that $T = MAC_{K_2}(C \| P_2 \| L_2)$. If the authentication tag is incorrect, output "invalid" and stop.
12) Output the message *M*.

Conformance region recommendation: A conformance region should include:

— At least one valid set of domain parameters
— At least one valid private key *s* for each set of domain parameters
— All valid public keys *v* associated with the set of domain parameters; if key validation is performed or a -DHC primitive is used, invalid public keys that are appropriately handled by the implementation may also be included in the conformance region
— A range of encrypted messages *C*, authentication tags *T*, key derivation parameters P_1, and encoding parameters P_2 (where the ranges of key derivation parameters and encoding parameters may each include only the empty string)

NOTES

1—This scheme is amenable to "single-pass" processing in DHAES mode. In non-DHAES mode, single-pass processing is possible if the method for encrypting the message is a key derivation function combined with a symmetric encryption scheme, but not if the method is a stream cipher based on a key derivation function, because the MAC key in that method may not be available until after the message has been encrypted. As the latter method is only recommended for relatively short messages (see D.5.3.2.2), this is not likely to be a disadvantage in practice.

2—This scheme, in the EC case with appropriate restrictions on the scheme options and inputs, may be compatible with techniques in ANSI X9.63 [B12].

11.4 IFES-EPOC

IFES-EPOC is Integer Factorization Encryption Scheme, EPOC version. It is based on the work of Fujisaki and Okamoto [FO99b].

Figure 12 illustrates the scheme.

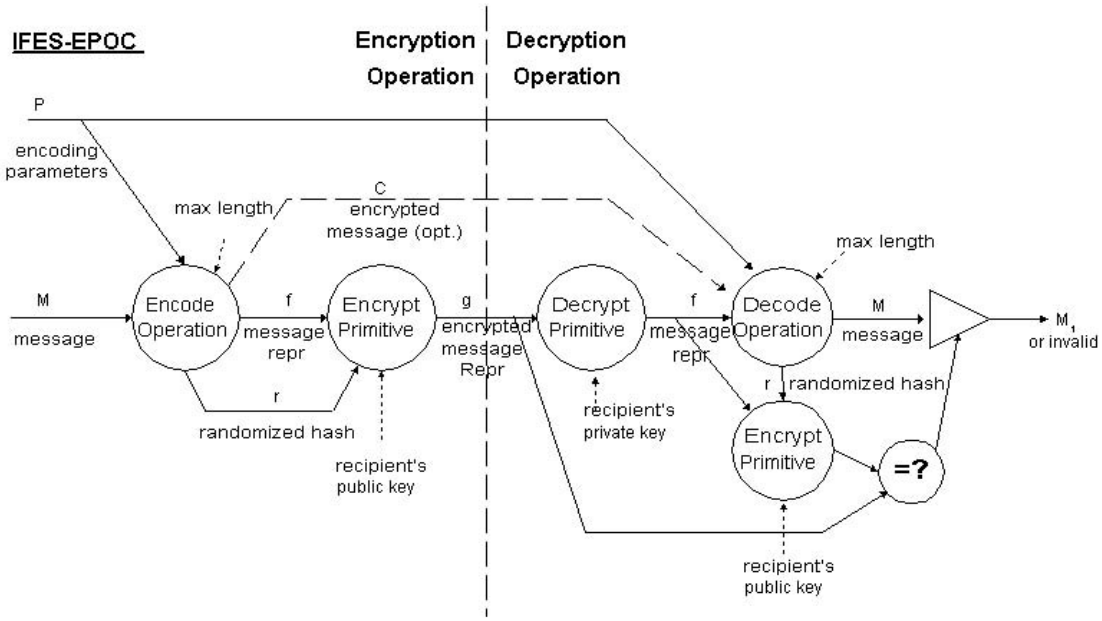

Figure 12—IFES-EPOC encryption and decryption operations

11.4.1 Scheme options

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

a) The encryption and decryption primitives, which shall be the pair IFEP-OU and IFDP-OU

b) The message-encoding method for encryption, which should be EME2 (see 12.2.2), EME3 (see 12.2.3), or a technique designated for use with IFES-EPOC in an amendment to this standard

c) The length in bits, *rBits*, of the randomized hash value (see D.5.3.2.1)

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

NOTES

1—The form of the ciphertext below depends on whether the selected message-encoding method outputs an encrypted message *C*. If it does, the ciphertext shall consist of an encrypted message representative *g* and the encrypted message *C*. If not, the ciphertext shall consist only of an encrypted message represented. EME3 outputs an encrypted message *C*, but EME2 does not.

2—The length in bits of the randomized hash value may depend on the public key.

11.4.2 Encryption operation

The ciphertext (g, C) (where C is optional) shall be generated by a sender from a message M and encoding parameters P by the following or an equivalent sequence of steps:

1) Obtain the recipient's purported public key (n, u, v, k) for the operation.
2) *(Optional)* Validate the public key (n, u, v, k). Stop if validation fails.
3) Set the maximum length of the message representative to be $l = k - 1$.
4) Use the encoding operation of the selected message-encoding method (see 11.4.1) to produce a message representative f of maximum length l, a randomized hash value r of maximum length *rBits*, and optionally an encrypted message C from the message M and the encoding parameters P (f and r will be non-negative integers). If the message is too long, the encoding method may not be able to produce a representative of the selected length. In that case, output "error" and stop.
5) Compute the encrypted message representative g from f, r, and the recipient's public key with the primitive IFEP-OU.
6) Output the pair (g, C) as the ciphertext.

Conformance region recommendation: A conformance region should include:

— At least one valid OU public key (n, u, v, k); if key validation is performed, invalid public keys that are appropriately handled by the implementation may also be included in the conformance region.
— A range of messages M and encoding parameters P (where the range of encoding parameters may include only the empty string)

11.4.3 Decryption operation

The message M shall be recovered from the ciphertext (g, C) (where C is optional) and encoding parameters P, by the following or an equivalent sequence of steps:

1) Select the private key (p, q, k, w) for the operation and obtain the corresponding public key (n, u, v, k).
2) Compute an integer f from g and the selected private key (p, q, k, w) with the primitive IFDP-OU.
3) Set the maximum length of the message representative to be $l = k - 1$, where n is the modulus in the IF private key.
4) Decode the integer f and optional encrypted message C if present according to the decoding operation of the selected message-encoding method (see 11.4.1), maximum length l, randomized hash length *rBits*, and the encoding parameters P to produce the message M and the randomized hash value r. If the decoding operation produces an error, output "invalid" and stop.
5) Compute another encrypted message representative g' from f, r, and the public key (n, u, v, k) with the primitive IFEP-OU.
6) If $g = g'$, output the message M. Otherwise, output "invalid."

Conformance region recommendation: A conformance region should include:

— At least one valid OU private key (p, q, k, w)
— All encrypted message representatives g in the range $[0, n - 1]$
— A range of encrypted messages C and encoding parameters P (where the range of encrypted messages is optional, depending on the encoding method, and the range of encoding parameters may include only the empty string)

NOTE—A potentially more efficient implementation of step 5) and step 6) is described in A.2.11.

12. Message-encoding methods

Replace the second and third paragraphs with the following text:

Each message-encoding method consists of an encoding operation, and of either a decoding or a verification operation. In general, an encoding operation encodes the message to produce a non-negative integer, called a *message representative*. It takes the message, optionally the maximum bit length *l* of the output, and possibly other parameters as input, and produces the integer of bit length no more than *l* (see 3.1 for the definition of bit length of an integer). A decoding operation gives back the original message (or part of it) given the message representative, optionally the length *l*, and the same additional parameters as were passed to the encoding operation. A verification operation verifies whether the message representative is a valid encoding of a message given the message representative, the message, optionally the length *l*, and the same parameters as were passed to the encoding operation. (The one exception to the model here is EMSR2, which operates on octet string message representatives.)

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

Other encoding methods are allowed within the framework of the schemes. As an example, an implementation may select an encoding method similar to one of those defined here, but based on a different underlying hash function or symmetric encryption scheme. Such an implementation may still claim conformance with the scheme (see Annex B), but with respect to the selected encoding method. The selection of encoding methods not recommended here requires further security analysis by the implementer.

12.1 Message-encoding methods for signatures with appendix

Insert the following note:

NOTE—See D.5.2.2.1 for security considerations related to these encoding methods.

12.1.1 EMSA1

Replace the second paragraph and the note with the following text:

The method is parameterized by the following choice:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSA1 in an amendment to this standard

NOTES

1—EMSA1 can handle messages of essentially any length in octets.

2—EMSA1 is amenable to “single-pass” processing in the typical case that the signature is transmitted after the message, because the message *M* can be processed by the verification operation before the signature is available.

12.1.1.1 Encoding operation

Replace the first input with the following text:

— A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)

Replace step 1) with the following text:

1) If the length of *M* is greater than the input length limitation for the selected hash function, output "error" and stop.

12.1.1.2 Verification operation

Replace the first input with the following text:

— A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)

12.1.2 EMSA2

Replace the second paragraph and the note with the following text:

The method is parameterized by the following choice:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSA2 in an amendment to this standard

The encoding method uses a hash function identifier, *HashID*, which is a single octet. The values of *HashID* for the hash functions in 14.1 are given in the table below. (See Note 3 after the table for an explanation of how the identifiers were assigned.)

Hash function	***HashID***
SHA-1 (14.1.1)	33
RIPEMD-160 (14.1.2)	31
SHA-256 (14.1.3)	34
SHA-384 (14.1.4)	36
SHA-512 (14.1.5)	35

NOTES

1—EMSA2 can handle messages of essentially any length in octets.

2—EMSA2 is amenable to "single-pass" processing in the typical case that the signature is transmitted after the message, because the message *M* can be processed by the verification operation before the signature is available.

3—*HashID* corresponds to the part of ISO/IEC 10118 in which the hash function is defined and the algorithm number within that part.[6] For instance, RIPEMD-160 is defined in ISO/IEC 10118-3:1998 [ISO-10118-3] and is hash function 1 within that part, so its hash function identifier is hexadecimal 31. The hash function identifiers for SHA-256, SHA-384, and SHA-512 were taken from a final revised draft of that document.

4—EMSA2 has a similar construction to that for signatures giving message recovery in ISO/IEC 9796-2:1997, in the common case that the length of the message representative is congruent to 7 mod 8 and the hash output is an octet string. In other cases, a more general construction based on ISO/IEC 9796-2:1997 may be appropriate.

Replace 12.1.2.1 with the following text:

12.1.2.1 Encoding operation

Input:

- A message, which is an octet string M (depending on the hash function chosen, there may be a length limitation)
- The maximum bit length l of the message representative (l shall be at least $8hLen + 31$ and shall be congruent to 7 mod 8)

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

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

1) If the length of M is greater than the input length limitation for the selected hash function, or $l < 8hLen + 31$, or $l + 1$ is not divisible by 8, output "error" and stop.
2) Compute $Hash(M)$ with the selected hash function to produce an octet string H of length $hLen$ octets.
3) If M is of length 0 (i.e., an empty string), let P_1 be a single octet with hexadecimal value 4b. Otherwise let P_1 be a single octet with hexadecimal value 6b.
4) Let P_2 be an octet string of length $\lfloor (l + 1)/8 \rfloor - hLen - 4$ octets, each with hexadecimal value bb.
5) Let P_3 be a single octet with value *HashID* (see the table in 12.1.2).
6) Let $T = P_1 \| P_2 \| ba \| H \| P_3 \| cc$, where ba and cc are single octets represented in hexadecimal.
7) Convert T to an integer f using OS2IP.
8) Output f as the message representative.

Replace 12.1.2.2 with the following text:

12.1.2.2 Verification operation

Input:

- A message, which is an octet string M (depending on the hash function chosen, there may be a length limitation)
- The maximum bit length l of the message representative (l shall be at least $8hLen + 31$ and shall be congruent to 7 mod 8)
- The message representative, which is an integer $f \geq 0$

Output: "Valid" if f is a correct representative of M; "invalid" otherwise

[6]Information on references can be found in Clause 2.

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

1) If the length of M is greater than the input length limitation for the selected hash function, or $l < 8hLen + 31$ or $l + 1$ is not divisible by 8, output "invalid" and stop.
2) Convert f to an octet string T of length $\lfloor (l + 1)/8 \rfloor$ octets using the primitive I2OSP.
3) If M is of length 0 (i.e., an empty string), let P_1 be a single octet with hexadecimal value 4b. Otherwise let P_1 be a single octet with hexadecimal value 6b.
4) Verify that the leftmost octet of T is equal to P_1, the next $\lfloor (l + 1)/8 \rfloor - hLen - 4$ octets each have hexadecimal value bb, and the next octet after that has hexadecimal value ba. If not, output "invalid" and stop.
5) Verify that the second rightmost octet has value *HashID* (see the table in 12.1.2). If not, output "invalid" and stop.
6) Verify that the rightmost octet has hexadecimal value cc. If not, output "invalid" and stop.
7) Remove the leftmost $\lfloor (l + 1)/8 \rfloor - 22$ octets of T and the rightmost 2 octets of T to produce an octet string H' of length $hLen$ octets.
8) Apply the selected hash function to M to produce an octet string H of length $hLen$ octets.
9) If $H = H'$ output "valid." Otherwise, output "invalid."

NOTE—As the encoding operation for EMSA3 is deterministic, the verification operation may equivalently be implemented by applying the encoding operation again to the message and comparing its output with the message representative.

Insert the following two subclauses (12.1.3 and 12.1.4) after 12.1.2:

12.1.3 EMSA3

EMSA3 is an encoding method for signatures with appendix based on a hash function with some additional formatting, based on PKCS #1 v1.5 [B126] (see also Kaliski and Staddon [KS98] and PKCS #1 v2.1 [PKCS1v2_1]). It is recommended for use with IFSSA (see 10.3).

The method is parameterized by the following choice:

— A hash function *Hash* without output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSA3 in an amendment to this standard

The encoding method uses a hash function identifier, *HashID*, which is an octet string. The values of *HashID* for the hash functions in 14.1 are given in the table below. (See Note 3 after the table for an explanation of how the identifiers were assigned.)

Hash function	***HashID***
SHA-1 (14.1.1)	30 21 30 09 06 05 2b 0e 03 02 1a 05 00 04 14
RIPEMD-160 (14.1.2)	30 21 30 09 06 05 2b 24 03 02 01 05 00 04 14
SHA-256 (14.1.3)	30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20
SHA-384 (14.1.4)	30 41 30 0d 06 09 60 86 48 01 65 03 04 02 02 05 00 04 30
SHA-512 (14.1.5)	30 51 30 0d 06 09 60 86 48 01 65 03 04 02 03 05 00 04 40

NOTES

1—EMSA3 can handle messages of essentially any length in octets.

2—EMSA3 is amenable to "single-pass" processing in the typical case that the signature is transmitted after the message, because the message *M* can be processed by the verification operation before the signature is available.

3—*HashID* corresponds to the portion of the DER encoding an ASN.1 value of type DigestInfo (see PKCS #1 v1.5 [B126]) that precedes the hash value itself. The type DigestInfo is defined as

```
DigestInfo ::= SEQUENCE {
 digestAlgorithm AlgorithmIdentifier, -- hash function algorithm identifier
 digest OCTET STRING -- hash value }
```

HashID thus consists of the tag and length fields for the SEQUENCE, the encoding of the AlgorithmIdentifier component (see ITU-T Recommendation X.509 [B82]), and the tag and length fields of the encoding of the digest component. If a hash function has a fixed-length output, the *HashID* value will be the same for all hash values for that hash function and can thus be considered a constant value, as is the case in the table in this subclause.

12.1.3.1 Encoding operation

Input:

- A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)
- The maximum bit length *l* of the message representative (*l* shall be at least 8*hashIDLen* + 8*hLen* + 80)

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

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

1) If the length of *M* is greater than the input length limitation for the selected hash function, output "error" and stop.
2) Let *HashID* be the identifier for the hash function (see the table in 12.1.3), and let *hashIDLen* be the length in octets of *HashID*.
3) Let $oLen = \lfloor l/8 \rfloor$. If *oLen* < *hashIDLen* + *hLen* + 10, output "error" and stop.
4) Compute *Hash* (*M*) with the selected hash function to produce an octet string *H* of length *hLen* octets.
5) Let *P* be an octet string of length *oLen* – *hashIDLen* – *hLen* – 2 octets, each with hexadecimal value ff. The length of *P* will be at least 8 octets.
6) Let $T = 01 \| P \| 00 \| HashID \| H$, where 01 and 00 are single octets represented in hexadecimal.
7) Convert *T* to an integer *f* using OS2IP.
8) Output *f* as the message representative.

12.1.3.2 Verification operation

Input:

- A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)
- The maximum bit length *l* of the message representative (*l* shall be at least 8*hashIDLen* + 8*hLen* + 80)
- The message representative, which is an integer $f \geq 0$

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

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

1) If the length of *M* is greater than the input length limitation for the selected hash function, output "invalid" and stop.
2) Let *HashID* be the identifier for the hash function (see the table in 12.1.3), and let *hashIDLen* be the length in octets of *HashID*.
3) Let $oLen = \lfloor l/8 \rfloor$. If *oLen* < *hashIDLen* + *hLen* + 10, output "invalid" and stop.
4) Convert *f* to an octet string *T* of length *oLen* octets using the primitive I2OSP. If the primitive outputs "error," output "invalid" and stop.
5) Verify that the leftmost octet of *T* is equal to 01, the next *oLen* – *hashIDLen* – *hLen* – 2 octets each have hexadecimal value ff, the next octet after that has hexadecimal value 00, and the next *hashIDLen* octets match *HashID*. If not, output "invalid" and stop.
6) Let *H*' be the rightmost *hLen* octets of *T*.
7) Apply the selected hash function to *M* to produce an octet string *H* of length *hLen* octets.
8) If *H* = *H*' output "valid." Otherwise, output "invalid."

NOTE—As the encoding operation for EMSA3 is deterministic, the verification operation may equivalently be implemented by applying the encoding operation again to the message and comparing its output with the message representative.

12.1.4 EMSA4

EMSA4 is an encoding method for signatures with appendix based on a hash function and a mask generation function, based on the Probabilistic Signature Scheme (PSS) (see Bellare and Rogaway [B19]). In one mode of operation, it is similar to Full-Domain Hashing (see Bellare and Rogaway [BR93]). It is recommended for use with IFSSA (see 10.3).

The method is parameterized by the following choices:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSA4 in an amendment to this standard
— A mask generation function *G*, which shall be MGF1 (see 14.2.1) with the same underlying hash function *Hash*, or a technique designated for use with EMSA4 in an amendment to this standard
— An indication as to whether hash function identification is desired
— A salt length in bits *saltBits*, a nonnegative integer; typical values are 8*hLen* and 0 (in which case the signature scheme is deterministic, similar to Full-Domain Hashing)

NOTES

1—EMSA4 can handle messages of essentially any length in octets.

2—EMSA4 is amenable to "single-pass" processing in the typical case that the signature is transmitted after the message, because the message *M* can be processed by the verification operation before the signature is available.

3—EMSA4 is a special case of EMSR3 where the recoverable part of the message is the empty string.

12.1.4.1 Encoding operation

Input:

— A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)
— The maximum bit length *l* of the message representative (*l* shall be at least 8*hLen* + *saltBits* + 8*u* + 1, where *u* = 2 if hash function identification is desired and *u* = 1 if not)

Output: A message representative, which is an integer $f \geq 0$ of bit length at most l where $f \equiv 12 \pmod{16}$; or "error"

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

1) Apply the EMSR3 encoding operation with the selected hash function, mask generation function, and salt length, and the indication as to whether hash function identification is desired, where the recoverable message part is the empty string, the nonrecoverable message part is M, and the maximum bit length of the message representative is l to compute a message representative f.
2) Output f as the message representative.

12.1.4.2 Verification operation

Input:

— A message, which is an octet string M (depending on the hash function chosen, there may be a length limitation)
— The maximum bit length l of the message representative (l shall be at least $8hLen + saltBits + 8u + 1$, where $u = 2$ if hash function identification is desired and $u = 1$ if not)
— The message representative, which is an integer $f \geq 0$

Output: "Valid" if f is a correct representative of M; "invalid" otherwise

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

1) Apply the EMSR3 decoding operation with the selected hash function, mask generation function, and salt length, and the indication as to whether hash function identification is desired, where the nonrecoverable message part is M, the maximum bit length of the message representative is l, and the message representative is f to recover a recoverable message part M_1. If the decoding operation outputs "invalid," output "invalid" and stop.
2) If M_1 is the empty string, output "valid." Otherwise, output "invalid."

12.1.5 EMSA5

EMSA5 is an encoding method for signatures with appendix based on a hash function and a mask generation function, based on the method in TSH-ESIGN (see Fujisaki and Okamoto [FO98]). It is recommended for use with IFSSA (see 10.6).

The method is parameterized by the following choices:

— A hash function *Hash* with output length *hBits* bits, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSA5 in an amendment to this standard
— A mask generation function *G*, which shall be MGF1 (see 14.2.1) with the same underlying hash function *Hash*, or a technique designated for use with EMSA5 in an amendment to this standard

NOTES

1—EMSA5 can handle messages of essentially any length in octets.

2—EMSA5 is amenable to "single-pass" processing in the typical case that the signature is transmitted after the message, because the message M can be processed by the verification operation before the signature is available.

12.1.5.1 Encoding operation

Input:

— A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)
— The maximum bit length *l* of the message representative

Output: A message representative, which is an integer $f \geq 0$; or "error"

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

1) If the length of *M* is greater than the input length limitation for the selected hash function, output "error" and stop.
2) Let $oLen = \lceil l/8 \rceil$ and let $t = 8oLen - l$. (Note that *t* will be between 0 and 7.)
3) Compute $H = Hash(M)$ with the selected hash function to produce an octet string *H* of length *hLen* octets.
4) Apply the mask generation function *G* to the string *H* to produce an octet string *T* of length *oLen* octets.
5) Set the leftmost *t* bits of *T* to zero.
6) Convert *T* to an integer *f* with the primitive OS2IP.
7) Output *f* as the message representative.

12.1.5.2 Verification operation

Input:

— A message, which is an octet string *M* (depending on the hash function chosen, there may be a length limitation)
— The maximum bit length *l* of the message representative
— The message representative, which is an integer $f \geq 0$ of bit length at most *l*

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

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

1) Use the Encoding Operation of EMSA5 to compute a message representative *g*, a non-negative integer. If the encoding operation outputs "error," output "invalid" and stop.
2) If $f = g$, output "valid." Else, output "invalid."

12.2 Message-encoding methods for encryption

Insert the following note:

NOTE—See D.5.3.2.1 for security considerations related to these encoding methods.

12.2.1 EME1

Replace the second paragraph with the following text:

The method is parameterized by the following choice:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EME1 in an amendment to this standard
— A mask generation function *G*, which shall be MGF1, or a technique designated for use with EME1 in an amendment to this standard

NOTE—EME1 can handle messages of length up to $\lfloor (l/8) - 2hLen - 1 \rfloor$ octets where *l* is the maximum length of the message representative.

12.2.1.1 Encoding operation

Replace the input list with the following text:

— The maximum bit length *l* of the message representative (*l* shall be at least 16*hLen* + 8)
— A message, which is an octet string *M* of length $mLen \leq (l/8) - 2hLen - 1$ octets
— Encoding parameters, which is an octet string *P* (*P* may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may be a length limitation)

Replace step 1) with the following text:

1) If the length of *P* is greater than the input length limitation for the selected hash function, output "error" and stop.

Insert the following note:

NOTE—Step 5) may be precomputed, if *P* is known in advance.

12.2.1.2 Decoding operation

Replace the input list with the following text:

— The maximum bit length *l* of the message representative (*l* shall be at least 16*hLen* + 8)
— The message representative, which is an integer $f \geq 0$
— Encoding parameters, which is an octet string *P* (*P* may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may be a length limitation)

Replace the steps with the following text:

1) If the length of *P* is greater than the input length limitation for the selected hash function, output "error" and stop.
2) Let $oLen = \lfloor l/8 \rfloor$ and *seedLen* = *hLen*. If *oLen* < *seedLen* + *hLen* + 1, output "error" and stop.
3) Apply the hash function *Hash* to the parameters *P* to produce an octet string *cHash* of length *hLen.*
4) Convert *f* to an octet string *EM* of length *oLen* + 1 with the primitive I2OSP. If the primitive outputs "error," output "error" and stop.

5) Let *X* be the leftmost octet of *EM*, let *maskedSeed* be the next *seedLen* octets, and let *maskedDB* be the remaining octets.
6) Apply the mask generation function *G* to the string *maskedDB* to produce an octet string *seedMask* of length *seedLen* octets.
7) Let *seed* = *maskedSeed* ⊕ *seedMask*.
8) Apply the mask generation function *G* to the string *seed* to produce an octet string *dbMask* of length *oLen* – *seedLen* octets.
9) Let *DB* = *maskedDB* ⊕ *dbMask*.
10) Let *M*' be all but the first *hLen* octets of *DB*.
11) Let *T* be the leftmost non-zero octet of *M*'.
12) Compare the first *hLen* octets of *DB* to *cHash*. If they are not equal, or if *X* ≠ hexadecimal value 00, or if *T* ≠ hexadecimal value 01, output "error" and stop. (The same error message should be output for each situation, at the same time—see Note 1.)
13) Remove *T* and all octets to the left of it (which are all zero) from *M*' to produce an octet string *M*.
14) Output the message *M*.

Insert the following notes:

NOTES

1—It is important that the implementation output the same error message at the same time in step 12) regardless of the cause of error. In particular, an implementation should not reveal whether the error is due to the value *X* being incorrect. Otherwise an opponent will be able to determine whether the most significant octet of an RSA decryption is nonzero, which may lead to an attack (see Manger [Man01] and D.5.3.2.1 Note 1 for further discussion).

2—The description of the decoding operation has been modified from that in IEEE Std 1363-2000 to make it clearer how to avoid the attack described by Manger [Man01]. This does not affect compatibility; the differences only affect when errors are detected. In particular, the error checks involving the message representative (except for the one in step 4), which does not affect security—see Note 3) are all done at step 12). In the previous description, the checks were distributed through the decoding operation, making it less evident that the same error message should be output at the same time in each case.

As stated in B.1, conformance with may be achieved by implementing either "identical" or "equivalent" steps to those specified. An implementation of a scheme that follows the steps specified for EME1 in IEEE Std 1363-2000, yet does not reveal information distinguishing which error occurred, may thus claim conformance with EME1 as specified here on the basis of "equivalence."

3—The error in step 4) cannot occur when the EME1 decoding operation follows the IFDP-RSA primitive as specified in IFES, because the message representative *f* output by IFDP-RSA is at most *l* + 1 bits long and the octet string *EM* is at least *l* + 1 bits long.

4—Step 3) may be precomputed, if *P* is known in advance.

Insert the following two subclauses (12.2.2 and 12.2.3) after 12.2.1:

12.2.2 EME2

EME2 is an encoding method for encryption based on the conversion in Fujisaki and Okamoto [FO99b], a hash function, and a mask generation function. It is recommended for use with IFES-EPOC (see 11.4). It can produce message representatives of arbitrary length *l*; however, there are restrictions on the length of the message it can encode that depend on *l*.

The method is parameterized by the following choices:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EME2 in an amendment to this standard

— A mask generation function *G*, which shall be MGF1 (see 14.2.1) with the same underlying hash function *Hash*, or a technique designated for use with EME2 in an amendment to this standard

— A seed length in octets *seedLen*, a non-negative integer; a typical value is *hLen*

NOTE—EME2 can handle messages of length up to $\lfloor (l + 7)/8 \rfloor - seedLen - 1$ octets where l is the maximum length of the message representative.

12.2.2.1 Encoding operation

Input:

— The maximum bit length l of the message representative (l shall be at least $8seedLen + 1$); the maximum bit length *rBits* of the randomized hash value

— A message, which is an octet string M of length $mLen \le \lfloor (l + 7)/8 \rfloor - seedLen - 1$ octets

— Encoding parameters, which is an octet string P (P may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may be a length limitation)

Output: A message representative, which is an integer $f \ge 0$ of bit length at most l, and a randomized hash value, which is an integer $r \ge 0$ of bit length at most $8hLen$; or "error"

The message representative f and randomized hash value r shall be computed by the following or an equivalent sequence of steps:

1) If the length of P plus $\lfloor (l + 7)/8 \rfloor$ is greater than the input length limitation for the selected hash function, output "error" and stop.
2) Let $oLen = \lfloor (l + 7)/8 \rfloor$. If $mLen > oLen - seedLen - 1$, then output "error" and stop. Let $rLen = \lceil (rBits)/8 \rceil$, and let $t = 8rLen - rBits$. (Note that t will be between 0 and 7.)
3) Let S be the octet string that consists of $oLen - mLen - seedLen - 1$ zero octets. Let T be the octet string consisting of a single octet with hexadecimal value 01.
4) Let $M' = S \| T \| M$.
5) Generate a fresh, random octet string *seed* of length *seedLen* octets (for more on generation of random strings, see D.6).
6) Let $EM = M' \| seed$.
7) Let $DB = EM \| P$.
8) Apply the hash function *Hash* to *DB* to produce an octet string H of length *hLen* octets.
9) Apply the mask generation function G to the string H to produce an octet string T of length *rLen* octets.
10) Set the leftmost t bits of T to zero.
11) Convert EM to an integer f with the primitive OS2IP.
12) Convert T to an integer r with the primitive OS2IP.
13) Output f as the message representative and r as the randomized hash value.

12.2.2.2 Decoding operation

Input:

— The maximum bit length l of the message representative (l shall be at least $8seedLen + 1$); the maximum bit length *rBits* of the randomized hash value
— The message representative, which is an integer $f \geq 0$
— Encoding parameters, which is an octet string P (P may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may a length limitation)

Output: The message, which is an octet string M of length at most $\lfloor l/8 \rfloor - seedLen - 1$ octets, and the randomized hash value, which is an integer $r \geq 0$ of bit length at most $8hLen$; or "error"

The message shall be decoded and the randomized hash value obtained by the following or an equivalent sequence of steps:

1) If the length of P plus $\lfloor l/8 \rfloor$ is greater than the input length limitation for the selected hash function, output "error" and stop.
2) Let $oLen = \lfloor l/8 \rfloor$. If $oLen < seedLen + hLen + 1$, output "error" and stop. Let $rLen = \lceil (rBits)/8 \rceil$, and let $t = 8rLen - rBits$. (Note that t will be between 0 and 7.)
3) Convert f to an octet string EM of length $oLen$ octets with the primitive I2OSP. If the primitive outputs "error," output "error" and stop.
4) Let M' be all but the rightmost $seedLen$ octets of EM.
5) Let T be the leftmost nonzero octet of M'. If $T \neq$ hexadecimal value 01, output "error" and stop.
6) Remove T and all octets to the left of it (which are all zero) from M' to produce an octet string M.
7) Let $DB = EM \| P$.
8) Apply the hash function *Hash* to DB to produce an octet string H of length $hLen$ octets.
9) Apply the mask generation function G to the string H to produce an octet string T of length $rLen$ octets.
10) Set the leftmost t bits of T to zero.
11) Convert T to an integer r with the primitive OS2IP.
12) Output the message M and the randomized hash r.

12.2.3 EME3

EME3 is another encoding method for encryption based on the conversion in Fujisaki and Okamoto [FO99b], a hash function, and a mask generation function. It is recommended for use with IFES-EPOC (see 11.4). It can produce message representatives of arbitrary length l. In contrast to EME2, the length of the message that it can encode is either unrestricted or constrained by a very large number.

The method is parameterized by the following choices:

a) A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EME3 in an amendment to this standard
b) A mask generation function *G*, which shall be MGF1 (see 14.2.1) with the same underlying hash function *Hash*, or a technique designated for use with EME3 in an amendment to this standard
c) The method for combining the seed with the message, which shall be either:
    1) A stream cipher based on a key derivation function, where the key derivation function shall be KDF2 (see 13.2) or a technique designated for use with EME3 in an amendment to this standard (this method is only recommended for relatively short messages such as symmetric keys—see D.5.3.2.1)

2) A key derivation function combined with a symmetric encryption scheme, where the key derivation function shall be KDF2 (see 13.2) or a technique designated for use with EME3 in an amendment to this standard, and the symmetric encryption scheme shall be one of the schemes in 14.3, or a technique designated for use with EME3 in an amendment to this standard

NOTES

1—EME3 can handle messages of essentially any length in octets.

2—EME3 is not amenable to "single-pass" processing because the entire message *M* must be processed by the hash function before the encrypted message *C* is processed.

3—The encoding method is also implicitly parameterized by any key derivation parameters for the key derivation function. As a default, the key derivation parameters should be the empty string. Parameters to the symmetric encryption scheme (if applicable) such as an initialization vector are local to that scheme.

12.2.3.1 Encoding operation

Input:

- The maximum bit length *l* of the message representative; the maximum bit length *rBits* of the randomized hash value
- A message, which is an octet string *M* of length *mLen* octets (depending on the hash function and mask generation function or symmetric encryption scheme chosen, there may be a length limitation)
- Encoding parameters, which is an octet string *P* of length *pLen* octets (*P* may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may be a length limitation)

Output: A message representative, which is an integer $f \geq 0$ of bit length at most *l*, a randomized hash value, which is an integer $r \geq 0$ of bit length at most *rBits*, and an encrypted message *C*, which is an octet string; or "error"

The message representative *f*, randomized hash value *r*, and encrypted message *C* shall be computed by the following or an equivalent sequence of steps:

1) If *mLen* is greater than the output length limitation for the selected key derivation function or the plaintext length limitation for the selected symmetric encryption scheme (whichever is applicable), output "error" and stop.
2) Let $seedLen = \lfloor l/8 \rfloor$. Let $rLen = \lceil (rBits)/8 \rceil$, and let $t = 8rLen - rBits$. (Note that *t* will be between 0 and 7.)
3) Generate a fresh, random octet string *seed* of length *seedLen* octets (for more on generation of random strings, see D.6).
4) If the method for combining the seed with the message is a stream cipher based on a key derivation function:
   a) Apply the selected key derivation function to *seed* to produce an octet string *mask* of length *mLen* octets.
   b) Let $C = M \oplus mask$.
5) If the method for combining the seed with the message is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a key *K* for the selected symmetric encryption scheme from the seed with the selected key derivation function (see 12.2.3, Note 3).
   b) Encrypt *M* under the key *K* with the selected symmetric encryption scheme to produce the encrypted message *C*.
6) Let $DB = M \,||\, seed \,||\, C \,||\, P$. If the length of *DB* is greater than the input length limitation for the selected hash function, output "error" and stop.

7) Apply the hash function *Hash* to *DB* to produce an octet string *H* of length *hLen* octets.
8) Apply the mask generation function *G* to the string *H* to produce an octet string *T* of length *rLen* octets.
9) Set the leftmost *t* bits of *T* to zero.
10) Convert *seed* to an integer *f* with the primitive OS2IP.
11) Convert *T* to an integer *r* with the primitive OS2IP.
12) Output *f* as the message representative, *r* as the randomized hash value, and *C* as the encrypted message.

12.2.3.2 Decoding operation

Input:

— The maximum bit length *l* of the message representative; the maximum bit length *rBits* of the randomized hash value
— The message representative, which is an integer $f \geq 0$
— The encrypted message, which is an octet string *C* of length *cLen* octets (depending on the hash function and mask generation function or symmetric encryption scheme chosen, there may be a length limitation)
— Encoding parameters, which is an octet string *P* of length *pLen* octets (*P* may be an empty string); see D.5.3.3 for possible uses (depending on the hash function chosen, there may a length limitation)

Output: The message, which is an octet string *M*, and the randomized hash value, which is an integer $r \geq 0$ of bit length at most 8*hLen*; or "error"

The message shall be decoded and the randomized hash value obtained by the following or an equivalent sequence of steps:

1) If *cLen* is greater than the output length limitation for the selected key derivation function or the ciphertext length limitation for the selected symmetric encryption scheme (whichever is applicable), output "error" and stop.
2) Let *seedLen* = $\lfloor l/8 \rfloor$. Let *rLen* = $\lceil ((rBits)/8) \rceil$, and let $t = 8rLen - rBits$. (Note that *t* will be between 0 and 7.)
3) Convert *f* to an octet string *seed* of length *seedLen* octets with the primitive I2OSP. If the primitive outputs "error," output "error" and stop.
4) If the method for combining the message with the seed is a stream cipher based on a key derivation function:
   a) Apply the selected key derivation function to *seed* to produce an octet string *mask* of length *cLen* octets.
   b) Let $M = C \oplus mask$.
5) If the method for combining the message with the seed is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a key *K* for the selected symmetric encryption scheme from the seed with the selected key derivation function (see 12.2.3, Note 3).
   b) Decrypt *C* under the key *K* with the selected symmetric encryption scheme to recover *M*. (If the decryption operation outputs "error," output "invalid" and stop.)
6) Let $DB = M \| seed \| C \| P$. If the length of *DB* is greater than the input length limitation for the selected hash function, output "error" and stop.
7) Apply the hash function *Hash* to *DB* to produce an octet string *H* of length *hLen* octets.
8) Apply the mask generation function *G* to the string *H* to produce an octet string *T* of length *rLen* octets.
9) Set the leftmost *t* bits of *T* to zero.
10) Convert *T* to an integer *r* with the primitive OS2IP.
11) Output the message *M* and the randomized hash *r*.

Insert the following subclause after 12.2:

12.3 Message-encoding methods for signatures giving message recovery

This standard strongly recommends the use of one of the following message-encoding methods for signature schemes giving message recovery.

NOTE—See D.5.2.2.2 for security considerations related to these encoding methods.

12.3.1 EMSR1

EMSR1 is an encoding method for signatures giving message recovery based on a hash function, based on ISO/IEC 9796-3:2000 [B79]. It is recommended for use with DL/ECSSR (see 10.4).

The method is parameterized by the following choices:

- A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSR1 in an amendment to this standard
- An indication as to whether hash function identification is desired
- The short redundancy length in octets *r1Len* and the long redundancy length in octets *r2Len*, each of which shall be at most *hLen* + 1 if hash function identification is desired, and at most *hLen* otherwise
- The encoding method optionally uses a hash function identifier, *HashID*, which is a single octet; the values of *HashID* are given in the table in 12.1.2

NOTES

1—EMSR1 can handle recoverable message parts of length up to $\lfloor l/8 \rfloor - r1Len$ octets if the nonrecoverable message part is empty, and $m1Len \leq \lfloor l/8 \rfloor - r2Len$ octets otherwise, where l is the maximum length of the message representative; the nonrecoverable message part can have essentially any length.

2—EMSR1 is not amenable to "single-pass" processing, because the nonrecoverable message part M_2 cannot be processed by the decoding operation until the signature has been processed to recover the recoverable message part M_1.

3—If hash function identification is desired, then *r1Len* and *r2Len* should be set to *hashLen* + 1 so that the hash function identifier is not truncated during the encoding operation. However, see D.5.2.2 for further discussion on hash function identification.

4—For compatibility with ISO/IEC 9796-3:2000, the length of the recoverable message part should be $\lfloor l/8 \rfloor - r2Len$ octets if the nonrecoverable message part is nonempty.

12.3.1.1 Encoding operation

Input:

- The maximum bit length l of the message representative (l shall be at least $8r1Len$ if the nonrecoverable message part is empty, or $8r2Len$ bits otherwise
- A recoverable message part, which is an octet string M_1 of length *m1Len* octets, where $m1Len \leq \lfloor l/8 \rfloor - r1Len$ if the nonrecoverable message part is empty, and $m1Len \leq \lfloor l/8 \rfloor - r2Len$ if the nonrecoverable message part is not empty
- A nonrecoverable message part, which is an octet string M_2 of length *m2Len* octets (depending on the hash function chosen, there may be a length limitation)
- A pre-signature, which is a bit string I

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

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

1) If the 8*m1Len* + 8*m2Len* plus the length in bits of *I* plus 128 is greater than the length limitation for the selected hash function, output "error" and stop.
2) If *m2Len* = 0, then let *rLen* = *r1Len*. Otherwise, let *rLen* = *r2Len*.
3) If *m1Len* > $\lfloor l/8 \rfloor$ – *rLen*, then output "error" and stop.
4) Convert *m1Len* to an octet string C_1 of length 8 octets and *m2Len* to an octet string C_2 of length 8 octets using I2OSP.
5) Let *M*' = OS2BSP (C_1 || C_2 || M_1 || M_2) || *I*.
6) Compute *Hash* (*M*') with the selected hash function to produce an octet string *H* of length *hLen* octets.
7) If hash function identification is desired, then let *H* = *H* || *HashID*, where *HashID* is a single octet identifying the selected hash function (see the table in 12.1.2).
8) Let *R* be the leftmost *rLen* octets of *H*.
9) Let *T* = *R* || M_1.
10) Convert *T* to an integer *f* with the primitive OS2IP.
11) Output *f* as the message representative.

12.3.1.2 Decoding operation

Input:

— The maximum bit length *l* of the message representative (*l* shall be at least 8*r1Len* if the nonrecoverable message part is empty, or 8*r2Len* bits otherwise)
— The message representative, which is an integer *f* ≥ 0 of bit length at most *l*
— The length in octets *m1Len* of the recoverable message part
— The nonrecoverable message part, which is an octet string M_2 of length *m2Len* octets (depending on the hash function chosen, there may be a length limitation)
— The pre-signature, which is a bit string *I*

Output: The recoverable message part M_1 or "invalid"

The recoverable message part shall be recovered by the following or an equivalent sequence of steps:

1) If 8*m1Len* + 8*m2Len* plus the length in bits of *I* plus 128 is greater than the length limitation for the selected hash function, output "invalid" and stop.
2) If *m2Len* = 0, then let *rLen* = *r1Len*. Otherwise, let *rLen* = *r2Len*.
3) If *m1Len* > $\lfloor l/8 \rfloor$ – *rLen*, then output "invalid" and stop.
4) Convert *f* to an octet string *T* of length *rLen* + *m1Len* octets with the primitive I2OSP. If the primitive outputs "error," output "invalid" and stop.
5) Let *R* be the leftmost *rLen* octets of *T*, and let M_1 be the rightmost *m1Len* octets.
6) Convert *m1Len* to an octet string C_1 of length 8 octets and *m2Len* to an octet string C_2 of length 8 octets using I2OSP.
7) Let *M*' = OS2BSP (C_1 || C_2 || M_1 || M_2) || *I*.
8) Compute *Hash* (*M*') with the selected hash function to produce an octet string *H*' of length *hLen* octets.
9) If hash function identification is desired, then let *H*' = *H*' || *HashID*, where *HashID* is a single octet identifying the selected hash function (see the table in 12.1.2).
10) Let *R*' be the leftmost *rLen* octets of *H*'.
11) If *R* = *R*' output the message part M_1. Otherwise, output "invalid."

12.3.2 EMSR2

EMSR2 is an encoding method for signatures giving message recovery based on simple message padding and randomization. It is recommended for use with DL/ECSSR-PV (see 10.5).

The method is parameterized by the following choices:

a) An integer *padLen* between 1 and 255 inclusive
b) The method for combining the pre-signature with the (padded) recoverable message part, which shall be either:
   1) A stream cipher based on a key derivation function, where the key derivation function shall be KDF2 (see 13.2) or a technique designated for use with EMSR2 in an amendment to this standard
   2) A key derivation function combined with a symmetric encryption scheme, where the key derivation function shall be KDF2 (see 13.2) or a technique designated for use with EMSR2 in an amendment to this standard, and the symmetric encryption scheme shall be one of the schemes in 14.3, or a technique designated for use with EMSR2 in an amendment to this standard

NOTES

1—EMSR2 can handle recoverable message parts of essentially any length in octets. The nonrecoverable message parts are not processed by EMSR2.

2—EMSR2 is, in principle, amenable to "single-pass" processing because the nonrecoverable message part is not processed at all by EMSR2 (and in fact DL/ECSSR-PV, which employs EMSR2, supports single-pass processing of the nonrecoverable message part if the message representative *C* is transmitted *before* the nonrecoverable message part).

3—The encoding method is also implicitly parameterized by any key derivation parameters for the key derivation function. As a default, the key derivation parameter should be the empty string. Parameters to the symmetric encryption scheme (if applicable) such as an initialization vector are local to that scheme.

12.3.2.1 Encoding operation

Input:

— A recoverable message part, which is an octet string M_1 of length *m1Len* octets (depending on the key derivation function or symmetric encryption scheme chosen, there may be a length limitation)
— A pre-signature, which is an octet string *I*

Output: A message representative, which is an octet string *C* of length *m1Len* + *padLen* octets

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

1) If *mLen* is greater than the output length limitation for the selected mask generation function or the plaintext length limitation for the selected symmetric encryption scheme (whichever is applicable), output "error" and stop.
2) Convert *padLen* to an octet string P_1 of length *1* with primitive I2OSP.
3) Let P_2 be the octet string formed from the octet P_1 repeated *padLen* times. (Thus, P_2 will have length *padLen*.)
4) Let $T = P_2 \| M_1$.
5) If the method for combining the pre-signature is a stream cipher based on a key derivation function:
   a) Apply the selected key derivation function to the pre-signature *I* to produce an octet string mask of length *m1Len* + *padLen* octets.
   b) Let $C = T \oplus mask$.

6) If the method for combining the pre-signature is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a key *K* for the selected symmetric encryption scheme from the pre-signature *I* with the key derivation function (see 12.3.2, Note 3).
   b) Encrypt *T* under the key *K* with the selected symmetric encryption scheme to produce the message representative *C*.
7) Output *C* as the message representative.

NOTE—The padding octet string P_2 is 01 if *padLen* = 1, 02 02 if *padLen* = 2, 03 03 03 if *padLen* = 3, 04 04 04 04 if *padLen* = 4, 05 05 05 05 05 if *padLen* = 5, and so on.

12.3.2.2 Decoding operation

Input:

— The message representative, which is an octet string *C* of length *cLen* octets (*cLen* shall be at least *padLen*; depending on the key derivation function or symmetric encryption scheme chosen, there may be an additional length limitation)
— The pre-signature, which is an octet string *I*

Output: The recoverable message part M_1 or "invalid"

The recoverable message part shall be recovered by the following or an equivalent sequence of steps:

1) If *cLen* is greater than the output length limitation for the selected mask generation function or the ciphertext length limitation for the selected symmetric encryption scheme (whichever is applicable), output "error" and stop.
2) If the method for combining the pre-signature is a stream cipher based on a key derivation function:
   a) Apply the selected key derivation function to the pre-signature *I* to produce an octet string *mask* of length *cLen* octets.
   b) Let $T = C \oplus mask$.
3 If the method for combining the pre-signature is a key derivation function combined with a symmetric encryption scheme:
   a) Derive a key *K* for the selected symmetric encryption scheme from the pre-signature *I* with the selected key derivation function (see 12.3.2, Note 3).
   b) Decrypt *C* under the key *K* with the selected symmetric encryption scheme to recover *T*. (If the decryption operation outputs "error," output "invalid" and stop.)
4) Let *tLen* be the length of *T* in octets. If $tLen < padLen$, output "invalid" and stop.
5) Let P_1 be the leftmost octet of *T*.
6) Convert P_1 to an integer *padLen*' with OS2IP.
7) If $padLen' \neq padLen$, output "invalid" and stop.
8) If $padLen \geq 2$, verify that the next $padLen - 1$ octets after the first octet of *T* are equal to the octet P_1. If not, output "invalid" and stop.
9) Let M_1 be the rightmost $tLen - padLen$ octets of *T*.
10) Output the octet string M_1.

12.3.3 EMSR3

EMSR3 is an encoding method for signatures giving message recovery based on a hash function and a mask generation function, based on the Probabilistic Signature Scheme with Recovery (PSS-R) (see Bellare and Rogaway [B19]). It is recommended for use with IFSSR (see 10.6).

The method is parameterized by the following choices:

— A hash function *Hash* with output length *hBits* bits, which shall be one of the hash functions in 14.1 or a technique designated for use with EMSR3 in an amendment to this standard
— A mask generation function *G*, which shall be MGF1 (see 14.2.1) with the same underlying hash function *Hash*, or a technique designated for use with EMSR3 in an amendment to this standard
— An indication as to whether hash function identification is desired
— A salt length in bits *saltBits*, a nonnegative integer; typical values are *hBits* and 0 (in which case the signature scheme is deterministic)

The encoding method optionally uses a hash function identifier, *HashID*, which is a single octet. The values of *HashID* are given in the table in 12.1.2.

NOTES

1—EMSR3 can handle recoverable message parts of length up to $l - saltBits - hBits - 8u - 1$ bits, where l is the maximum length of the message representative and where $u = 2$ if hash function identification is desired and $u = 1$ if not; the nonrecoverable message part can have essentially any length. Note that for consistency with the draft revision to ISO/IEC 9796-2, the value of l should be at least $saltBits + hBits + 8u + 8$ so that EMSR3 can handle recoverable message parts of length up to at least 7 bits, and thus messages of any total length in bits. (This constraint will be satisfied for typical values of l, *saltBits* and *hBits*.)

2—EMSR3 is amenable to "single-pass" processing in the case that the signature is transmitted after the nonrecoverable message part, because the nonrecoverable message part M_2 can be processed by the decoding operation before the signature is available.

3—EMSR3 is defined in terms of bit strings for consistency with the draft revision to ISO/IEC 9796-2. However, if the message parts, salt, and hash output are all octet strings, then the encoding and decoding operations can be implemented with octet string operations only (except for setting certain bits to 0 or 1). Note that as invoked in this standard (i.e., by IFSSR), the recoverable message part for EMSR3 is always an octet string.

4—The dependence of the mask generation function on the underlying hash function helps protect against weak hash function attacks, as further discussed in D.5.2.2.

12.3.3.1 Encoding operation

Input:

— The maximum bit length l of the message representative (l shall be at least $saltBits + hBits + 8u + 1$, where $u = 2$ if hash function identification is desired and $u = 1$ if not)
— A recoverable message part, which is an octet string M_1 of length $m1Len \leq \lfloor (1 - saltBits - hBits - 8u - 1/(8)) \rfloor$ octets or a bit string MB_1 of length $m1Bits \leq l - saltBits - hBits - 8u - 1$ bits
— A nonrecoverable message part, which is an octet string M_2 of length *m2Len* octets (depending on the hash function chosen, there may be a length limitation)

Output: A message representative, which is an integer $f \geq 0$ of bit length at most l where $f \equiv 12 \pmod{16}$; or "error"

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

1) If the recoverable message part is provided as an octet string M_1, then convert it to a bit string MB_1 of length $8m1Len$ bits with the primitive OS2BSP. Let $m1Bits = 8m1Len$. Let $u = 2$ if hash function identification is desired and $u = 1$ if not.
2) If the length of M_2 is greater than the input length limitation for the selected hash function or if $m1Bits > l - saltBits - hBits - 8u - 1$, output "error" and stop.

3) Compute *Hash* (M_2) with the selected hash function to produce a bit string H_2 of length *hBits* bits.
4) Let $oLen = \lceil l/8 \rceil$, and let $t = 8oLen - l$. (Note that t will be between 0 and 7.)
5) Generate a fresh, random bit string *salt* of length *saltBits* bits (for more on generation of random strings, see D.6). (If *saltLen* is 0, then *salt* will simply be the empty string.)
6) Convert *m1Bits* to a bit string C_1 of length 64 bits using I2BSP.
7) Let $M' = C_1 \| MB_1 \| H_2 \| salt$.
8) Compute *Hash* (M') with the selected hash function to produce a bit string H of length *hBits* bits.
9) Let S be the bit string that consists of $8oLen - m1Bits - saltBits - hBits - 8u - 1$, 0 bits.
10) Let $DB = S \| 1 \| MB_1 \| salt$, where 1 is a single bit. The length of DB is $8oLen - hBits - 8u$ bits.
11) Apply the mask generation function G to the string H to produce a bit string *dbMask* of length $8oLen - hBits - 8u$ bits.
12) Let $maskedDB = DB \oplus dbMask$.
13) Set the leftmost t bits of *maskedDB* to zero.
14) If hash function identification is desired, then let $T = maskedDB \| H \| HashID \| cc$, where *HashID* is a single octet identifying the selected hash function (see the table in 12.1.2) and where *cc* is a single octet represented in hexadecimal. If not, then let $T = maskedDB \| H \| bc$, where *bc* is a single octet represented in hexadecimal. The length of T is $8oLen$ bits, which is a multiple of 8, but only the rightmost $8oLen - t = l$ bits may be nonzero.
15) Convert T to an integer f with the primitive OS2IP.
16) Output f as the message representative.

NOTES

1—If a random number generator is not available in step 4), a fixed value may be selected for *salt* with a resulting security analysis similar to that for Full-Domain Hashing (see Bellare and Rogaway [BR93]). Alternatively, *saltBits* may be specified as 0 to avoid the need for random number generation. See D.5.2.2 for further discussion.

2—The security analysis for EMSR3 addresses both the case that the nonrecoverable message part M_2 is input to the module computing the signature and the case that the hash of the nonrecoverable message part, Hash (M_2), is input. Thus, this hash operation may be performed outside the module without loss in the security analysis [which is not the case for the original version of PSS (see Bellare and Rogaway [B19])].

12.3.3.2 Decoding operation

Input:

— The maximum bit length l of the message representative (l shall be at least $saltBits + hBits + 8u + 1$, where $u = 2$ if hash function identification is desired and $u = 1$ if not)
— The message representative, which is an integer $f \geq 0$ of bit length at most l
— The nonrecoverable message part, which is an octet string M_2 of length *m2Len* octets (depending on the hash function chosen, there may be a length limitation)

Output: The recoverable message part, which is an octet string M_1 of length at most $\lfloor (1 - saltBits - hBits - 8u - 1)/8 \rfloor$ octets or a bit string MB_1 of length at most $l - saltBits - hBits - 8u - 1$ bits; or "invalid"

The recoverable message part shall be recovered by the following or an equivalent sequence of steps:

1) Let $u = 2$ if hash function identification is desired and $u = 1$ if not. If the length of M_2 is greater than the input length limitation for the selected hash function or if $l < saltBits + hBits + 8u + 1$, output "invalid" and stop.
2) Compute *Hash* (M_2) with the selected hash function to produce a bit string H_2 of length *hBits* bits.

3) Let $oLen = \lceil l/8 \rceil$, and let $t = 8oLen - l$. (Note that t will be between 0 and 7.)

4) Convert f to an octet string T of length $oLen$ octets with the primitive I2OSP. If the primitive outputs "error," output "invalid" and stop.

5) If hash function identification is desired, then verify that the rightmost octet of T has hexadecimal value cc and that the second rightmost octet of T is *HashID*, where *HashID* is a single octet identifying the selected hash function (see the table in 12.1.2). If hash function identification is not desired, then verify that the rightmost octet of T has hexadecimal value bc. In either case, if verification fails, output "invalid" and stop.

6) Let *maskedDB* be the leftmost $8oLen - hBits - 8u$ bits of T, and let H be the next *hBits* bits.

7) Verify that the leftmost t bits of *maskedDB* are zero.

8) Apply the mask generation function G to the string H to produce a bit string *dbMask* of length $8oLen - hBits - 8u$ bits.

9) Let $DB = maskedDB \oplus dbMask$.

10) Set the leftmost t bits of DB to zero.

11) Let MB_1' be the leftmost $8oLen - saltBits - hBits - 8u$ bits of DB, and let *salt* be the rightmost *saltBits* bits.

12) Let U be the leftmost nonzero bit of MB_1'. If MB_1' has no nonzero bits, output "invalid" and stop.

13) Remove U and all bits to the left of it (which are all zero) from MB_1' to produce a bit string MB_1. (The bit string MB_1 may be empty.) Let *m1Bits* be the length in bits of MB_1.

14) Convert *m1Bits* to a bit string C_1 of length 64 bits using I2BSP.

15) Let $M' = C_1 \| MB_1 \| H_2 \| salt$.

16) Compute *Hash* (M') with the selected hash function to produce an octet string H' of length *hBits* bits.

17) If $H \neq H'$ output "invalid."

18) If the output is to be returned as a bit string, output MB_1 as the recoverable message part. Otherwise, convert MB_1 to an octet string M_1 with the primitive BS2OSP and output M_1 as the recoverable message part.

13. Key derivation functions

Replace the first two paragraphs with the following text:

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

The use of an inadequate key derivation function may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the key derivation functions contained in this clause, or any key derivation functions defined in subsequent addenda to this standard. Similar to the situation for encoding methods, other key derivation functions are allowed within the framework of the schemes (see the introduction to Clause 10 for further discussion). However, the selection of key derivation functions not recommended here requires further security analysis by the implementer.

Insert the following subclause after 13.1:

13.2 KDF2

KDF2 is based on the constructions in ANSI ~~X9.42:2001~~X9.42-2001 ([B8], Section 7.7.2) and ANSI X9.63 ([B12], Section 5.6.3). It is recommended for use with DL and EC key agreement schemes (Clause 9), the DL and EC encryption scheme (see 11.3), EMSR2 (see 12.3.2), and EME3 (see 12.2.3).

The function is parameterized by the following choice:

— A hash function *Hash* with output length *hBits* bits, which shall be one of the hash functions in 14.1 or a technique designated for use with KDF2 in an amendment to this standard

Input:

— A shared secret string, which is an octet string *Z* of length *zLen* octets or a bit string *ZB* of length *zBits* bits (depending on the hash function chosen, there may be a length limitation)

— The desired length in octets of the output, which is a positive integer *oLen*, or the desired length in bits of the output, which is a positive integer *oBits* [8*oLen* or *oBits* shall be less than or equal to $hBits \times (2^{32} - 1)$]

— Key derivation parameters, which is an octet string *P* of length *pLen* octets or a bit string *PB* of length *pBits* bits (depending on the hash function chosen, there may be a length limitation)

Output: A shared secret key, which is an octet string *K* of length *oLen* octets or a bit string *KB* of length *oBits* bits; or "error"

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

1) If the shared secret string is provided as an octet string *Z*, then convert it to a bit string *ZB* of length 8*zLen* bits with the primitive OS2BSP. Let *zBits* = 8*zLen*.
2) If the desired length of the output is provided as an octet length *oLen*, then let *oBits* = 8*oLen*.
3) If the key derivation parameters are provided as an octet string *P*, then convert it to a bit string *PB* of length 8*pLen* bits with the primitive OS2BSP. Let *pBits* = 8*pLen*.
4) If *zBits* + *pBits* + 32 is greater than the input length limitation for the selected hash function or if $oBits > hBits \times (2^{32} - 1)$, output "error" and stop.
5) Let *MB* be the empty string. Let $cThreshold = \lceil (oBits)/(hBits) \rceil$.
6) Let *counter* = 1.
   6.1) Convert *counter* to a bit string *CB* of length 32 bits using I2BSP.
   6.2) Compute *Hash* (*ZB* || *CB* || *PB*) with the selected hash function to produce a bit string *HB* of *hBits* bits.
   6.3) Let *MB* = *MB* || *HB*.
   6.4) Increment *counter* by one. If *counter* ≤ *cThreshold*, go to step 6.1).
7) Let *KB* be the leftmost *oBits* bits of *MB*.
8) If the output is to be returned as a bit string, output *KB* as the shared secret key. Otherwise, convert *KB* to an octet string *K* with the primitive BS2OSP and output *K* as the shared secret key.

NOTES

1—KDF2 can accept and return either octet strings or bit strings to support the varying requirements of techniques in IEEE Std 1363-2000 and this amendment. For convenience, octet string operands are converted to bit strings here, and only bit strings are passed to the underlying hash function. This conversion assumes that the underlying hash function would convert an octet string to a bit string the same way that KDF2 does, i.e., most significant bit first. Although this is true for the hash functions allowed by IEEE Std 1363-2000 and this amendment (SHA-1, SHA-256, SHA-384, SHA-512, and RIPEMD-160), it is not true in general for all hash functions. When the operands are all octet strings, KDF2 can be implemented with octet string operations only, skipping the conversion to bit strings and passing octet strings directly to the underlying hash function.

2—The interpretation and usage of the output *K* in general is beyond the scope of this standard (although within one of the encryption schemes and certain encoding methods, a particular interpretation and usage is specified).

3—As *counter* is only 32 bits and starts at 1, the maximum length of the output is at most $hBits \times (2^{32} - 1)$ bits.

4—This method is the same as MGF1 in the case that *P* is the empty string except that the counter starts at 1 rather than 0 and the output is a bit string rather than an octet string.

Replace the title of Clause 14 with the following:

14. Auxiliary techniques

Insert the following paragraph and note before 14.1:

This clause describes a number of additional non-public-key techniques supporting the methods in this standard. The list of techniques given here reflects generally recommended practice, and it is limited in the interest of encouraging interoperability. (See C.4 for rationale.) However, the list is not intended to be exhaustive. Other techniques are allowed within the framework of the schemes, although of course the selection of techniques not listed here requires further security analysis by the implementer.

NOTE—Most of the functions defined here can accept and return either octet strings or bit strings to support the varying requirements of techniques in IEEE Std 1363-2000 and this amendment. For convenience of specification, octet string operands are converted to bit strings via OS2BSP (i.e., most significant bit first), so that only bit strings are passed to the underlying hash function. Likewise, an octet string result is obtained via BS2OSP. These conversions assume that the underlying hash function would convert the same way. Although this is true for the hash functions allowed by IEEE Std 1363-2000 and this amendment (SHA-1, SHA-256, SHA-384, SHA-512, and RIPEMD-160), it is not true in general for all hash functions. When the operands are all octet strings, the functions can be implemented with octet string operations only.

14.1 Hash functions

Replace the first paragraph with the following text:

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

The use of an inadequate hash function may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the hash functions contained in this subclause, or any hash functions defined in subsequent addenda to this standard. However, as noted in the introduction to this clause, other hash functions are allowed within the framework of the schemes in this standard, with appropriate security analysis by the implementer.

Replace 14.1.1 with the following text:

14.1.1 SHA-1

SHA-1 is defined in FIPS PUB 180-2 [B55]. The hash output length for SHA-1 is 160 bits, and the block size (relevant to MAC1) is $B = 512$.

Input: An octet string *M* of length *mLen* octets (*mLen* has to be less than 2^{61}) or a bit string *MB* of length *mBits* bits (*mBits* has to be less than 2^{64})

Output: A hash value, which is an octet string *H* of length 20 octets or a bit string *HB* of length 160 bits

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

1) If the input is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2) Apply the Secure Hash Algorithm, revision 1, as described in FIPS PUB 180-2 to *MB* to produce a bit string *HB* of length 160 bits.
3) If the output is to be returned as a bit string, output *HB* as the hash value. Otherwise, convert *HB* to an octet string *H* with the primitive BS2OSP and output *H* as the hash value.

Replace 14.1.2 with the following text:

14.1.2 RIPEMD-160

RIPEMD-160 is based on the work of Dobbertin et al. [B49]. The hash output length for RIPEMD-160 is 160 bits, and the block size (relevant to MAC1) is $B = 512$.

Input: An octet string *M* of length *mLen* octets (*mLen* shall be at most $2^{61}-1$) or a bit string *MB* of length *mBits* bits (*mBits* shall be at most $2^{64}-1$)

Output: A hash value, which is an octet string *H* of length 20 octets or a bit string *HB* of length 160 bits

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

1) If the input is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2) Apply the RIPEMD-160 hash algorithm as described in ISO/IEC 10118-3:1998 [ISO-10118-3], clause 7 to *MB* to produce a bit string *HB* of length 160 bits.
3) If the output is to be returned as a bit string, output *HB* as the hash value. Otherwise, convert *HB* to an octet string *H* with the primitive BS2OSP and output *H* as the hash value.

Insert the following four subclauses (14.1.3–14.1.6) after 14.1.2:

14.1.3 SHA-256

SHA-256 is defined in FIPS PUB 180-2 [B55]. The hash output length for SHA-256 is 256 bits, and the block size (relevant to MAC1) is $B = 512$.

Input: An octet string *M* of length *mLen* octets (*mLen* shall be at most $2^{61}-1$) or a bit string *MB* of length *mBits* bits (*mBits* shall be at most $2^{64}-1$)

Output: A hash value, which is an octet string *H* of length 32 octets or a bit string *HB* of length 256 bits

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

1) If the input is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2) Apply the SHA-256 hash algorithm as described in FIPS PUB 180-2 to *MB* to produce a bit string *HB* of length 160 bits.
3) If the output is to be returned as a bit string, output *HB* as the hash value. Otherwise, convert *HB* to an octet string *H* with the primitive BS2OSP and output *H* as the hash value.

14.1.4 SHA-384

SHA-384 is defined in FIPS PUB 180-2 [B55]. The hash output length for SHA-384 is 384 bits, and the block size (relevant to MAC1) is $B = 1024$.

Input: An octet string *M* of length *mLen* octets (*mLen* shall be at most $2^{125}-1$) or a bit string *MB* of length *mBits* bits (*mBits* shall be at most $2^{128}-1$)

Output: A hash value, which is an octet string *H* of length 48 octets or a bit string *HB* of length 384 bits

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

1) If the input is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2) Apply the SHA-384 hash algorithm, as described in FIPS PUB 180-2 [B55], to *MB* to produce a bit string *HB* of length 384 bits.
3) If the output is to be returned as a bit string, output *HB* as the hash value. Otherwise, convert *HB* to an octet string *H* with the primitive BS2OSP and output *H* as the hash value.

14.1.5 SHA-512

SHA-256 is defined in FIPS PUB 180-2 [B55]. The hash output length for SHA-256 is 512 bits, and the block size (relevant to MAC1) is $B = 1024$.

Input: An octet string *M* of length *mLen* octets (*mLen* shall be at most $2^{125}-1$) or a bit string *MB* of length *mBits* bits (*mBits* shall be at most $2^{128}-1$)

Output: A hash value, which is an octet string *H* of length 64 octets or a bit string *HB* of length 512 bits

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

1) If the input is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP.
2) Apply the SHA-512 hash algorithm as described in FIPS PUB 180-2 [B55] to *MB* to produce a bit string *HB* of length 512 bits.
3) If the output is to be returned as a bit string, output *HB* as the hash value. Otherwise, convert *HB* to an octet string *H* with the primitive BS2OSP and output *H* as the hash value.

14.2 Mask generation functions

Replace the first paragraph with the following text:

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

The use of an inadequate mask generation function may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the mask generation functions contained in this subclause, or any mask generation functions defined in subsequent addenda to this standard. However, as noted in the introduction to this clause, other mask generation functions are allowed within the framework of the schemes in this standard, with appropriate security analysis by the implementer.

Replace 14.2.1 with the following text:

14.2.1 MGF1

MGF1 is a mask generation function based on a hash function, following the ideas of Bellare and Rogaway ([B18] and [B19]). It is used with EMSA4 (see 12.1.4), EME1 (see 12.2.1), and EMSR3 (see 12.3.3).

The function is parameterized by the following choice:

— A hash function *Hash* with output length *hBits* bits, which shall be one of the hash functions in 14.1 or a technique designated for use with MGF1 in an amendment to this standard

Input:

— An octet string *Z* of length *zLen* octets or a bit string *ZB* of length *zBits* bits (depending on the hash function chosen, there may be a length limitation)
— The desired length of the output, which is a positive integer *oLen* giving the length in octets if the output is to be returned as an octet string, a positive integer *oBits* giving the length in bits if the output is to be returned as a bit string (8*oLen* or *oBits* shall be at most $hBits \times 2^{32}$)

Output: A mask value, which is an octet string *mask* of length *oLen* octets or a bit string *maskB* of length *oBits* bits; or “error”

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

1) If the input is provided as an octet string *Z*, then convert it to a bit string *ZB* of length 8*zLen* bits with the primitive OS2BSP. Let *zBits* = 8*zLen*.
2) If the output is to be returned as an octet string (so that the desired length of the output is provided as an octet length *oLen*), then let *oBits* = 8*oLen*.
3) If *zBits* + 32 is greater than the input length limitation for the selected hash function, or if *oBits* > $hBits \times 2^{32}$, output “error” and stop.
4) Let *MB* be the empty string. Let *cThreshold* = $\lceil (oBits)/(hBits) \rceil$.
5) Let *counter* = 0.
   5.1) Convert *counter* to a bit string *CB* of length 32 bits using I2BSP.
   5.2) Compute *Hash* (*ZB* || *CB*) with the selected hash function to produce a bit string *HB* of *hBits* bits.
   5.3) Let *MB* = *MB* || *HB*.

5.4) Increment *counter* by one. If *counter* < *cThreshold*, go to step 5.1).

5.5) Let *maskB* be the leftmost *oBits* bits of *MB*.

6) If the output is to be returned as a bit string, output *maskB* as the mask value. Otherwise, convert *maskB* to an octet string *mask* with the primitive BS2OSP and output *mask* as the mask value.

NOTES

1—MGF1 can accept and return either octet strings or bit strings to support the varying requirements of techniques in IEEE Std 1363-2000 and this amendment. For convenience, octet string operands are converted to bit strings here, and only bit strings are passed to the underlying hash function. This conversion assumes that the underlying hash function would convert an octet string to a bit string the same way that MGF1 does, i.e., most significant bit first. Although this is true for the hash functions allowed by IEEE Std 1363-2000 and this amendment (SHA-1, SHA-256, SHA-384, SHA-512, and RIPEMD-160), it is not true in general for all hash functions. When the operands are all octet strings, MGF1 can be implemented with octet string operations only, skipping the conversion to bit strings and passing octet strings directly to the underlying hash function.

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

Insert the following two subclauses (14.3 and 14.4) after 14.2:

14.3 Symmetric encryption schemes

This subclause describes the symmetric encryption schemes used in this standard as building blocks for some encryption schemes and encoding methods. A symmetric encryption scheme consists of an encryption operation and a decryption operation. The encryption operation takes as input a message, which is an octet string or a bit string, and a key, and outputs an encrypted message, which is an octet string or a bit string. The encryption operation may be deterministic or randomized. The decryption operation takes as input an encrypted message and a key, and it outputs a message or "error."

The use of an inadequate symmetric encryption scheme may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the symmetric encryption schemes contained in this subclause, or any symmetric encryption schemes defined in subsequent addenda to this standard. However, as noted in the introduction to this clause, other symmetric encryption schemes are allowed within the framework of the (public-key) schemes in this standard, with appropriate security analysis by the implementer.

NOTE—The symmetric encryption schemes defined here both require that the length in bits of the message is divisible by 8. To encrypt messages whose length is not divisible by 8, a different symmetric encryption scheme should be chosen (perhaps one similar to those defined here, with slightly different padding; the specifics of such a choice are left to the implementer).

14.3.1 Triple-DES-CBC-IV0

Triple-DES-CBC-IV0 is two- or three-key triple-DES in Cipher Block Chaining (CBC) mode with a null initialization vector, as further specified in ANSI X9.52-1998 [B10], FIPS PUB 46-3 [FIPS-46-3], and FIPS PUB 113 [FIPS-113].

The underlying block cipher for the *Enc* and *Dec* functions below is the Data Encryption Algorithm (i.e., the DES algorithm from FIPS PUB 46-3 [FIPS-46-3]), which operates on a 64 bit key (of which every eighth bit is a parity bit) and a 64 bit block.

14.3.1.1 Encryption operation

Input:

— A key, which is a bit string *KB* of 128 or 192 bits (where every eighth bit of the key, starting at the left does not affect the operation and may be used for parity checking)
— A message, which is an octet string *M* of length *mLen* octets or a bit string *MB* of length *mBits* bits (where *mBits* shall be divisible by 8)

Output: A encrypted message, which is an octet string *C* of length *cLen* octets where $cLen = 8\lceil (mLen + 1)/8 \rceil$ or a bit string *CB* of length *cBits* bits, where $cBits = 64\lceil (mBits + 8)/64 \rceil$

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

1) Let K_1 be the leftmost 64 bits of *KB*, and let K_2 be the next 64 bits. If length of the bit string *KB* is 192 bits, then let K_3 be the rightmost 64 bits of *KB*. Otherwise let $K_3 = K_1$.
2) If the input is provided as a bit string *MB*, then convert it to an octet string *M* of length *mBits*/8 octets with the primitive BS2OSP. Let *mLen* = *mBits*/8.
3) Let *padLen* = 8 – (*mLen* mod 8), and let $k = \lceil (mLen + 1)/8 \rceil$.
4) Convert *padLen* to an octet string P_1 of length 1 with primitive I2OSP.
5) Let P_2 be the octet string formed from the octet P_1 repeated *padLen* times. (Thus, P_2 will have length *padLen*.)
6) Let $T = M \| P_2$.
7) Consider the padded message *T* as a sequence of eight-octet blocks $T_1, \ldots, T_k$ and the encrypted message *C* as a sequence of eight-octet blocks $C_1, \ldots, C_k$.
8) Let C_0 be an octet string that consists of eight zero octets. (C_0 is not part of the encrypted message.)
9) Let $i = 1$.
   9.1) Let $U = T_i \oplus C_{i-1}$.
   9.2) Let $C_i = Enc\ (K_3\ (Dec\ (K_2, Enc\ (K_1, U))))$ where *Enc* denotes encryption with the Data Encryption Algorithm and *Dec* denotes decryption.
   9.3) Increment *i* by one. If $i \le k$, go to step 9.1).
10) Let $C = C_1 \| C_2 \| \ldots \| C_k$.
11) If the output is to be returned as an octet string, output *C* as the encrypted message. Otherwise, convert *C* to a bit string *CB* of length 64*k* bits with the primitive OS2BSP and output *CB* as the encrypted message.

NOTE—The padding octet string P_2 is 01 if *padLen* = 1, 02 02 if *padLen* = 2, 03 03 03 if *padLen* = 3, 04 04 04 04 if *padLen* = 4, 05 05 05 05 05 if *padLen* = 5, and so on.

14.3.1.2 Decryption operation

Input:

— A key, which is a bit string *KB* of 128 or 192 bits (where every eighth bit of the key, starting at the left, does not affect the operation and may optionally be used for parity checking)
— An encrypted message, which is an octet string *C* of length *cLen* octets (*cLen* shall be a multiple of 8 and *cLen* shall be greater than or equal to 8) or a bit string *CB* of length *cBits* bits (*cBits* shall be a multiple of 64 and *cBits* shall be greater than or equal to 64)

Output: A message, which is an octet string *M* of length *mLen* octets where $cLen - 8 \le mLen \le cLen - 1$ or a bit string *MB* of length *mBits* bits, where $cBits - 64 \le mBits \le cBits - 8$; or "error"

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

1) If the input is provided as a bit string *CB*, then convert it to an octet string *C* of length *cBits*/8 octets with the primitive BS2OSP. Let *cLen* = *cBits*/8.
2) If *cLen* is not a multiple of 8, or *cLen* < 8, output "error" and stop.
3) Let K_1 be the leftmost 64 bits of *KB*, and let K_2 be the next 64 bits. If length of the bit string *KB* is 192 bits, then let K_3 be the rightmost 64 bits of *KB*. Otherwise let $K_3 = K_1$.
4) Let *k* = *cLen*/8.
5) Consider the encrypted message *C* as a sequence of eight-octet blocks $C_1, \ldots, C_k$ a padded message *T* as a sequence of eight-octet blocks $T_1, \ldots, T_k$.
6) Let C_0 be an octet string that consists of eight zero octets. (C_0 is not part of the encrypted message.)
7) Let *i* = 1.
   7.1) Let $U = Dec\ (K_1, Enc\ (K_2, Dec\ (K_3, C_i)))$, where *Dec* denotes decryption with the Data Encryption Algorithm and *Enc* denotes encryption.
   7.2) Let $T_i = U \oplus C_{i-1}$.
   7.3) Increment *i* by one. If $i \le k$, go to step 7.1).
8) Let P_1 be the rightmost octet of T_k.
9) Convert P_1 to an integer *padLen* with OS2IP.
10) If *padLen* < 1 or *padLen* > 8, output "error" and stop.
11) If *padLen* ≥ 2, verify that the next *padLen* – 1 octets before the last octet of T_k are equal to the octet P_1. If not, output "error" and stop.
12) Let *mLen* = *cLen* – *padLen*, and let *M* be the leftmost *mLen* octets of *T*.
13) If the output is to be returned as an octet string, output the octet string *M* as the message. Otherwise, convert *M* to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP and output *MB* as the message

14.3.2 AES-CBC-IV0

AES-CBC is the Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining (CBC) mode with a null initialization vector, as further specified in FIPS PUB 197 [FIPS-197] and FIPS PUB 113 [FIPS-113].

The underlying block cipher for the *Enc* and *Dec* functions below is the AES algorithm, which operates on a 128-bit, 192-bit, or 256-bit key and a 128-bit block.

14.3.2.1 Encryption operation

Input:

— A key, which is a bit string *KB* of length 128, 192, or 256 bits
— A message, which is an octet string *M* of length *mLen* octets or a bit string *MB* of length *mBits* bits (where *mBits* shall be divisible by 8)

Output: A encrypted message, which is an octet string *C* of length *cLen* octets where $cLen = 16\lceil (mLen + 1)/16 \rceil$ or a bit string *CB* of length *cBits* bits where $cBits = 128\lceil (mBits + 8)/128 \rceil$

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

1) If the input is provided as a bit string *MB*, then convert it to an octet string *M* of length *mBits*/8 octets with the primitive BS2OSP. Let *mLen* = *mBits*/8.
2) Let *padLen* = 16 – (*mLen* mod 16), and let $k = \lceil (mLen + 1)/16 \rceil$.
3) Convert *padLen* to an octet string P_1 of length 1 with primitive I2OSP.

4) Let P_2 be the octet string formed from the octet P_1 repeated *padLen* times. (Thus, P_2 will have length *padLen*.)
5) Let $T = M \| P_2$.
6) Consider the padded message T as a sequence of 16-octet blocks $T_1, \ldots, T_k$ and the encrypted message C as a sequence of 16-octet blocks $C_0, \ldots, C_k$.
7) Let C_0 be an octet string that consists of 16 zero octets. (C_0 is not part of the encrypted message.)
8) Let $i = 1$.
   8.1) Let $U = T_i \oplus C_{i-1}$.
   8.2) Let $C_i = Enc\ (KB, U)$ where *Enc* denotes encryption with the AES algorithm.
   8.3) Increment i by one. If $i \le k$, go to step 8.1).
9) Let $C = C_1 \| C_2 \| \ldots \| C_k$.
10) If the output is to be returned as an octet string, output C as the encrypted message. Otherwise, convert C to a bit string CB of length $128k$ bits with the primitive OS2BSP and output CB as the encrypted message.

14.3.2.2 Decryption operation

Input:

— A key, which is a bit string *KB* of length 128, 192, or 256 bits
— An encrypted message, which is an octet string C of length *cLen* octets (*cLen* shall be a multiple of 16 and *cLen* shall be greater than or equal to 16) or a bit string CB of length *cBits* bits (*cBits* shall be a multiple of 128 and *cBits* shall be greater than or equal to 128)

Output: A message, which is an octet string M of length *mLen* octets where $cLen - 16 \le mLen \le cLen - 1$ or a bit string MB of length *mBits* bits where $cBits - 128 \le mLen \le cBits - 8$; or "error"

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

1) If the input is provided as a bit string CB, then convert it to an octet string C of length $cBits/8$ octets with the primitive BS2OSP. Let $cLen = cBits/8$.
2) If *cLen* is not a multiple of 16, or $cLen < 16$, output "error" and stop.
3) Let $k = cLen/16$.
4) Consider the encrypted message C as a sequence of 16-octet blocks $C_0, \ldots, C_k$ a padded message T as a sequence of 16-octet blocks $T_1, \ldots, T_k$.
5) Let C_0 be an octet string that consists of 16 zero octets. (C_0 is not part of the encrypted message.)
6) Let $i = 1$.
   6.1) Let $U = Dec\ (KB, C_i)$ where *Dec* denotes decryption with the AES algorithm.
   6.2) Let $T_i = U \oplus C_{i-1}$.
   6.3) Increment i by one. If $i \le k$, go to step 6.1).
7) Let P_1 be the rightmost octet of T_k.
8) Convert P_1 to an integer *padLen* with OS2IP.
9) If $padLen < 1$ or $padLen > 16$, output "error" and stop.
10) If $padLen \ge 2$, verify that the next $padLen - 1$ octets before the last octet of T_k are equal to the octet P_1. If not, output "error" and stop.
11) Let $mLen = cLen - padLen$, and let M be the leftmost *mLen* octets of T.
12) If the output is to be returned as an octet string, output the octet string M as the message. Otherwise, convert M to a bit string MB of length $8mLen$ bits with the primitive OS2BSP and output MB as the message.

14.4 Message authentication codes

This subclause describes message authentication codes used in this standard as building blocks for some encryption schemes. A message authentication code computes a tag from a secret key and data.

The use of an inadequate message authentication code may compromise the security of the scheme in which it is used. This standard strongly recommends the use of the message authentication code contained in this subclause, or any message authentication codes defined in subsequent addenda to this standard. However, as noted in the introduction to this clause, other message authentication codes are allowed within the framework of the (public-key) schemes in this standard, with appropriate security analysis by the implementer.

14.4.1 MAC1

MAC1 is the HMAC message authentication code, which is based on the work of Bellare et al. [BCK96], as further specified in ANSI X9.71-2000 [ANSI-X9.71] and the forthcoming HMAC FIPS [FIPS-198]. (See also RFC 2104 by Krawcyzk et al. [KBC97].) The function is parameterized by the following choices:

— A hash function *Hash* with output length *hLen* octets, which shall be one of the hash functions in 14.1 or a technique designated for use with MAC1 in an amendment to this standard (*B* denotes the block length in octets of the hash function)
— The length in bits of the tag, which is a positive integer *tBits* (*tBits* shall be greater than or equal to 32 and less than or equal to *hBits*)

Input:

— A key, which is a bit string *KB* of length *kBits* bits (*kBits* shall be a multiple of 8 and *kBits*/8 shall be greater than or equal to *hLen*/2)
— A message, which is an octet string *M* of length *mLen* octets or a bit string *MB* of length *mBits* bits (depending on the hash function chosen, there may be a length limitation)

Output: A tag, which is a bit string *TB* of length *tBits* bits or (in the case that *tBits* is divisible by 8) an octet string *T* of length *tBits*/8 octets

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

1) If the message is provided as an octet string *M*, then convert it to a bit string *MB* of length 8*mLen* bits with the primitive OS2BSP. Let *mBits* = 8*mLen*.
2) If *kBits* > 8*B*, then compute *Hash* (*K*) with the selected hash function to produce an octet string *KK* of length *hLen* octets. Otherwise, convert *K* to an octet string *KK* of length *kBits*/8 octets with the primitive BS2OSP. Let *kkLen* be the length of *KK* in octets.
3) Let *P* be an octet string of length *B* – *kkLen* octets, each with hexadecimal value 00.
4) Let $K_0 = KK \| P$.
5) Let *iPad* be an octet string of length *B* octets, each with hexadecimal value 36, and let *oPad* be an octet string of length *B* octets, each with hexadecimal value 5c.
6) Compute *Hash* (OS2BSP ($K_0 \oplus iPad$) || *MB*) with the selected hash function to produce an octet string *H* of length *hLen* octets.
7) Compute *Hash* (($K_0 \oplus oPad$) || *H*) with the selected hash function to produce an octet string *HH* of length *hLen* octets.
8) If the output is to be returned as a bit string, convert *HH* to a bit string *HHB* of length *tBits* bits with the primitive OS2BSP and output the leftmost *tBits* bits of *HHB* as the tag. Otherwise, output the leftmost *tBits*/8 octets of *HH* as the tag.

NOTE—MAC1 can accept and return either octet strings or bit strings to support the varying requirements of techniques in this amendment. For convenience, octet string operands are converted to bit strings here, and only bit strings are passed to the underlying hash function. This conversion assumes that the underlying hash function would convert an octet string to a bit string the same way that MAC1 does, i.e., most significant bit first. Although this is true for the hash functions allowed by IEEE Std 1363-2000 and this amendment (SHA-1, SHA-256, SHA-384, SHA-512, and RIPEMD-160), it is not true in general for all hash functions. When the operands are all octet strings, MAC1 can be implemented with octet string operations only, skipping the conversion to bit strings and passing octet strings directly to the underlying hash function.

Annex A

(informative)

Number-theoretic algorithms

A.2 Integer and modular arithmetic: algorithms

Replace the title of A.2.9 with the following:

A.2.9 An implementation of IF signature primitives (RSA and RW cases)

Insert the following two subclauses (A.2.10 and A.2.11) after A.2.9:

A.2.10 Multiplication in (Z/p^2Z)

Let p be a prime. The following algorithm efficiently computes the product of two elements of (Z/p^2Z), as required in IFDP-OU. It may also be applied to IFSP-ESIGN.

Input: Two integers u and v modulo p^2, represented as $u = a + bp$ and $v = c + dp$, where $0 \le a, b, c, d < p$

Output: $w = uv \bmod p^2$, represented as $w = e + hp$

1) Set $e \leftarrow ac \bmod p$ and $f = \lfloor (ac)/p \rfloor$. (These values can be computed simultaneously using a standard division algorithm.)
2) Set $g \leftarrow (ad + bc) \bmod p$.
3) Set $h \leftarrow (f + g) \bmod p$.
4) Output $w = e + hp$.

To apply this algorithm to IFSP-ESIGN, observe that step 3) in IFSP-ESIGN requires the computation of $\exp(r, e) \bmod n$ where $n = p^2q$. This can be done by computing $\exp(r, e) \bmod p^2$ using the above algorithm for multiplication in (Z/p^2Z), computing $\exp(r, e)$ modulo q, and then combining the results using the Chinese Remainder Theorem similarly to A.2.9. This can be done more efficiently if components such as $p^{-2} \bmod q$ are precomputed.

A.2.11 An implementation of IFES-EOPC decryption

In order to verify the structure of a received ciphertext (g, C), the IFES-EPOC decryption operation in 11.4.3 re-encrypts the recovered message M and verifies that the resulting encrypted message representative g' matches the received ciphertext g.

Instead of calling the IFEP-OU encryption primitive to regenerate g' and comparing it to g, one can instead adopt the following more efficient verification procedure.

Suppose that randomized hash value is r and the message representative is f. The IFES-EPOC decryption operation checks whether the following equation holds:

$$g \equiv \exp(u,f) \times \exp(v, r) \bmod n$$

or equivalently whether the equation holds modulo p^2 and modulo q. However, as noted below, it is sufficient to check modulo p and q. Thus, step 5) and step 6) of the IFES-EPOC decryption operation may be replaced with the following:

> 5'. Check whether $g \equiv \exp(u, f) \times \exp(v, r) \bmod p$ and $g \equiv \exp(u, f) \times \exp(v, r) \bmod q$. If so, output the message M. Otherwise, output "invalid."

However, as observed in [NTT01], it is sufficient from a security perspective to check modulo q only. Thus, step 5) and 6) may be replaced with

> 5". Check whether $g \equiv \exp(u, f) \times \exp(v, r) \bmod q$. If so, output the message M. Otherwise, output "invalid."

This is the same as calling the IFEP-OU primitive with the "public" key (q, u, v, k) rather than (n, u, v, k), then checking that the resulting g' is the same as g. As a further improvement, one may reduce r modulo q-1 prior to exponentiation.

NOTE—The rationale for checking modulo p rather than modulo p^2 is based on the group isomorphism

$$(\phi, \phi):(Z/p^2Z) \rightarrow (Z/pZ) \times (Z/pZ)^*$$

where $\phi(x) = L(x_P)/L(u_p) \bmod p$ and $\phi'(x) = x \bmod p$. (Following 8.1.3.3, $x_P = \exp(x, p\text{-}1) \bmod p^2$ and $L(x_P) = (x_P - 1)/p \bmod p$.) The message representative f in the IFES-EPOC decryption operation is recovered by computing

$$f = L(g_n)/w \bmod p$$

where w is from the private key. Thus, the equation $\phi(g) = \phi(\exp(u, f) \times \exp(v, r) \bmod p^2) = \phi(\exp(u, f) \bmod p^2)$ already holds, and one only has to check whether $\phi'(g) = \phi'(\exp(u, f) \times \exp(v, r) \bmod p)$.

A.3 Binary finite fields: overview

A.3.1 Finite fields

Replace the last two paragraphs with the following text:

Finite fields exist for every prime-power order. This standard identifies three classes of finite fields that are commonly used in cryptography:

- When q is a prime p, the field $GF(p)$ is called a *prime finite field.* The field $GF(p)$ is typically represented as the set of integers modulo p. See A.1.2 for a discussion of these fields.
- When $q = 2^m$ for some m, the field $GF(2^m)$ is called a *binary finite field*. Unlike the prime field case, there are many common representations for binary finite fields. These fields are discussed below (A.3.3 to A.3.9).
- When $q = p^m$ for some $m > 1$ and some odd prime p, the field $GF(p^m)$ is called an *odd-characteristic extension field*. Elements of the field $GF(p^m)$ are typically represented by polynomials modulo an irreducible field polynomial. These fields are discussed further in A.17.

These are the three kinds of finite fields considered in this standard.

A.6.1 Checking for a normal basis

Change the first and last lines of step 1) to the following:

1) ~~Compute~~ Represent the successive squares of $t \pmod{p(t)}$ as

Delete the last line of step 1) ("modulo 2 via repeated squaring").

A.7 Basis conversion for binary fields

Insert the following four subclauses (A.7.5–A.7.8) after A.7.4:

A.7.5 Storage-efficient conversion techniques

The matrix multiplication technique described in A.7.1 requires one to store m^2 binary coefficients for the matrix Γ or Γ^{-1} and perform linear algebra for each conversion. The techniques below, which are based on work by Kaliski and Yin [KY98], require much smaller storage, after a setup operation that depends only on the choice of basis, at the expense of a few extra operations per conversion. They generally use field multiplication in the internal basis rather than linear algebra; if field multiplication in the internal basis is optimized, the performance of these algorithms is comparable with the matrix multiplication method. Although the setup operation for conversion in one direction is simpler than computing the change-of-basis matrix, the typical setup operation in the other direction is more complex, involving the computation of a multiplication matrix and a matrix inverse in addition to the change-of-basis matrix.

As in A.7.1, let B_1 be the basis in which the user performs field operations, and B_0 be the other basis. A.7.6 considers the task of importing (converting a representation in B_0 to a representation B_1) and A.7.8 that of exporting (converting a representation in B_1 to a representation in B_0). The two tasks are inherently different, because it is assumed that the user can efficiently perform field operations in B_1 but not in B_0.

Related methods for finite fields other than binary fields and representations other than polynomial and normal basis may be found in Kaliski and Yin [KY98] and Kaliski and Liskov [KL99].

A.7.6 Storage-efficient import

Input: A field degree m; a (polynomial or normal) basis B_0 with field polynomial $p_0(t)$; a bit string $\underline{a}$ of length m (Note: In the case of a Gaussian normal basis, the field polynomial can be computed via A.7.2.)

Output: The element $v \in GF(2^m)$ (represented in the internal basis B_1) such that the representation of v in B_0 is $\underline{a}$

If B_0 is a Polynomial Basis:

Setup (once per basis)

1) Let u be a root of $p_0(t)$ represented with respect to B_1. (u can be computed via A.5.6; see also Note 2 on p. 94.)

Conversion (per each element)

2) Let $\underline{a} = (a_{m-1} \ldots a_2\ a_1\ a_0)$.
3) Set $e \leftarrow 1$ (the element one of $GF(2^m)$, represented with respect to B_1).
4) If $a_{m-1} = 0$, then set $v \leftarrow 0$; else set $v \leftarrow e$.
5) For i from $m - 2$ downto 0 do
   5.1) Set $v \leftarrow vu$.
   5.2) If $a_i = 1$, set $v \leftarrow v + e$.
6) Output v.

If B_0 is a Normal Basis:

Setup (once per basis)

1) Let u be a root of $p_0(t)$ represented with respect to B_1. (u can be computed via A.5.6.)

Conversion (per each element)

2) Let $\underline{a} = (a_0\ a_1\ a_2 \ldots a_{m-1})$.
3) If $a_0 = 0$, then set $v \leftarrow 0$; else set $v \leftarrow u$.
4) For i from 1 to $m - 1$ do
   4.1) Set $v \leftarrow v^2$.
   4.2) If $a_i = 1$, set $v \leftarrow v + u$.
5) Output v.

Example (B_0 *is a Polynomial Basis*): Suppose, as in the second example in A.7.3, that B_0 is the polynomial basis modulo $p_0(u) = u^5 + u^4 + u^2 + u + 1$, and B_1 is the polynomial basis modulo $p_1(t) = t^5 + t^2 + 1$. Then a root of $p_0(u)$ is given by $u = t + 1$, and let this be the root selected in the setup operation.

Suppose $\underline{a} = (10001)$ is the bit string to be converted. Then the conversion operation sets $v \leftarrow 1$ in step 4), and in step 5) computes the following sequence of values:

i	a_i	v
3	0	$t + 1$
2	0	$t^2 + 1$
1	0	$t^3 + t^2 + t + 1$
0	1	t^4

The result, $v = t^4$, matches the bit string that would be computed by matrix multiplication:

$$(10001)\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (10000).$$

Example (B_0 *is a Normal Basis*): Suppose, as in the first example in A.7.6, that B_0 is the Type I optimal normal basis for $GF(2^4)$, and B_1 is the polynomial basis modulo $p_1(t) = t^4 + t + 1$. Then a root of $p_0(u) = u^4 + u^3 + u^2 + u + 1$ is given by $u = t^3 + t^2$; let this be the root selected in the setup operation.

Suppose $\underline{a}$ = (1001) is the bit string to be converted. Then the conversion operation sets $v \leftarrow t^3 + t^2$ in step 4), and in step 5) computes the following sequence of values:

i	a_i	v
1	0	$t^3 + t^2 + t + 1$
2	0	$t^3 + t$
3	1	t^2

The result, $v = t^2$, matches the bit string that would be computed by matrix multiplication:

$$(1001)\begin{bmatrix}1&1&0&0\\1&1&1&1\\1&0&1&0\\1&0&0&0\end{bmatrix} = (0100)$$

NOTES

1—Step 1) of the above algorithms needs be performed only once for each pair B_0 and B_1; the value u can then be stored and used every time an element needs to be imported. This is analogous to (but more space-efficient than) storing the matrix Γ to perform the import operation. The value u need not be kept secret, but it must be protected from modification to ensure the correctness of subsequent per-element conversion operations. (This is no different than for the matrix multiplication technique, where the matrix must be protected from modification but need not be kept secret.)

2—This algorithm may be invoked to facilitate division using the Extended Euclidean algorithm of A.4.4. Specifically, when B_1 is a normal basis, one needs to export elements α and β to a polynomial basis B_0 before computing $\gamma = \alpha / \beta$ via the Extended Euclidean algorithm, and then import γ back to B_1. In that case, the best choice for B_0 is the polynomial basis with polynomial $f(t)$, where $f(t)$ is the normal polynomial for B_1. Then, in step 1), u should not be computed via A.5.6; instead, u will simply be the element whose representation with respect to B_1 is (100…0). (Computing u via A.5.6 would require one to use division and could thus cause infinitely deep recursion if division is again implemented via basis conversion combined with the Extended Euclidean algorithm.)

A.7.7 Multiplication matrix for a polynomial basis

The export method presented in A.7.8 requires a computation of the (leftmost) multiplication matrix M for the basis B_1. For a polynomial basis, the matrix can be computed by the following algorithm. An algorithm for a normal basis is given in A.6.3.

Input: A field degree m; a polynomial basis B_1

Output: The matrix M such that m_{ij} is the leftmost bit of the product of the i-th and j-th basis vectors of B_1

1) Let t be the root of the polynomial defining B_1 [t is represented in B_1 by the m-bit string (00…010)].
2) Let $r := t^{m-1}$ [represented in B_1 by the m-bit string (100…0)].
3) Let $b_0 := b_1 := \ldots := b_{m-2} := 0$, and let $b_{m-1} := 1$.

4) For i from m to $2m - 2$ do
   4.1) Let $r := rt$.
   4.2) Let b_i be the leftmost bit of the representation of r in B_1.
5) Set

$$M \leftarrow \begin{bmatrix} b_{2m-2} & b_{2m-3} & \cdots & b_{m-1} \\ b_{2m-3} & b_{2m-4} & \cdots & b_{m-2} \\ . & . & \cdots & . \\ b_{m-1} & b_{m-2} & \cdots & b_0 \end{bmatrix}$$

6) Output M.

Example: Continuing the first example in A.7.6, let B_1 be the polynomial basis modulo $p_1(t) = t^5 + t^2 + 1$. Then the algorithm in step 4) computes:

i	r	b_i
5	$t^2 + 1$	0
6	$t^3 + t$	0
7	$t^4 + t^2$	1
8	$t^3 + t^2 + t + 1$	0

so the matrix M is

$$M := \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A.7.8 Storage-efficient export

Input: A field degree m; a (polynomial or normal) basis B_0 with field polynomial $p_0(t)$; an element $v \in GF(2^m)$ (represented in the internal basis B_1). (Note: In the case of a Gaussian normal basis, the field polynomial can be computed via A.7.2.)

Output: The bit string $\underline{a}$ of length m such that the representation of v in B_0 is $\underline{a}$

If B_0 is a Polynomial Basis:

Setup (once per basis)

1) Compute the matrix G via A.7.3. Let u be a root of $p_0(t)$ represented with respect to B_1. (It is computed in A.7.3.) Compute $w := u^{-1}$ via A.4.4. (See Note 3 on p. 99.)

2) Set $\Delta \leftarrow \Gamma^{-1}$, and let $\underline{s} = (\delta_{0m-1}\ \delta_{1m-1} \ldots \delta_{m-1m-1})$ be a row-vector whose elements are the $(m-1)$-st column of Δ.
3) Compute the multiplication matrix M for B_1 via A.6.3 or A.7.7 (note that A.6.3 includes a separate efficient method for computing the multiplication matrix in the case of a Gaussian normal basis using A.3.7).
4) Let $\underline{z} = s\ M^{-1}$. Note that $\underline{z}$ is an m-element bit string; let z be the element of $GF(2^m)$ whose representation in B_1 is $\underline{z}$.

Conversion (per each element)

5) Set $v \leftarrow vz$.
6) For $i = 0$ to $m-2$
   6.1) Let a_i be the leftmost bit of the representation of v in B_1.
   6.2) If $a_i = 1$, then set $v \leftarrow v + z$.
   6.3) Set $v := vw$.
7) Let a_{m-1} be the leftmost bit of the representation of v in B_1.
8) Output $\underline{a} := (a_{m-1} \ldots a_2\ a_1\ a_0)$.

If B_0 is a Normal Basis:

Setup (once per basis)

1) Compute the matrix Γ via A.7.3.
2) Set $\Delta \leftarrow \Gamma^{-1}$, and let $\underline{s} = (\delta_{0m-1}\ \delta_{1m-1} \ldots \delta_{m-1m-1})$ be a row-vector whose elements are the $(m-1)$-st column of Δ.
3) Compute the multiplication matrix M for B_1 via A.7.7.
4) Let $\underline{z} = \underline{s}\ M^{-1}$. Note that $\underline{z}$ is an m-element bit string; let z be the element of $GF(2^m)$ whose representation in B_1 is $\underline{z}$.

Conversion (per each element)

5) For $i = m-1$ downto 1
   5.1) Compute $w := vz$, and let a_i be the leftmost bit of the representation of w in B_1.
   5.2) Set $v := v^2$.
6) Compute $w := vz$, and let a_0 be the leftmost bit of the representation of w in B_1.
7) Output $\underline{a} := (a_0\ a_1\ a_2 \ldots a_{m-1})$.

Example (B_0 *is a Polynomial Basis*): Continuing the first example in A.7.6, let B_0 be the polynomial basis modulo $p_0(u) = u^5 + u^4 + u^2 + u + 1$ and let B_1 be the polynomial basis modulo $p_1(t) = t^5 + t^2 + 1$. Let the root of $p_0(u)$ selected be $u = t + 1$. The setup operation computes

$$\Gamma := \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$w := u^{-1} = t^4 + t^3 + t^2$$

$$\Delta := \Gamma^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\underline{s} = (11111)$

$$M := \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M^{-1} := \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$z = \underline{s}\, M^{-1} = (11100)$

$z = t^4 + t^3 + t^2$

(Note that $\Gamma = \Gamma^{-1}$ and $w = z$ in this example.)

Suppose $v = t^4$ is the field element to be converted. Then the conversion operation sets $v \leftarrow t^4 + t + 1$ in step 5), and in step 6) computes the following sequence of values:

i	a_i	***v***
0	1	$t^2 + 1$
1	0	$t + 1$
2	0	1
3	0	$t^4 + t^3 + t^2$

In step 7), the conversion operation computes $a^4 = 1$. The result is $\underline{a} = (10001)$.

Example (B_0 *is a Normal Basis*): Continuing the second example in A.7.6, let B_0 be the Type I optimal normal basis for GF(2^4) and let B_1 be the polynomial basis modulo $p_1(t) = t^4 + t + 1$. Let the root of $p_0(u)$ selected be $u = t^3 + t^2$. The setup operation computes

$$\Gamma := \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\Delta := \Gamma^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\underline{s} = (11111)$$

$$M := \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$M^{-1} := \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$z = \underline{s}\, M^{-1} = (1110)$$

$$z = t^3 + t^2 + t^1$$

Suppose $v = t^2$ is the field element to be converted. Then the conversion operation in step 5) computes the following sequence of values:

i	***w***	a_i	***v***
3	$t^3 + t^2 + t + 1$	1	$t + 1$
2	1	0	$t^2 + 1$
1	$t + 1$	0	t

In step 6), the conversion operation computes $w = t^3 + t^2 + t + 1$ and $a_0 = 1$. The result is $\underline{a} = (1001)$.

NOTES

1—Step 1)–Step 4) of the above algorithms need be performed only once for each pair B_0 and B_1; the value z (and w, in the case of polynomial basis) can then be stored and used every time an element needs to be exported. This is analogous to (but more space-efficient than) storing the matrix Γ^{-1} to perform the export operation. (Note that the matrix inversion can be avoided if Γ^{-1} is computed directly as the change-of-basis matrix from B_1 to B_0 via A.7.3.) Similarly to the import operation, the value z (and w, in the polynomial case) need not be kept secret but must be protected from modification.

2—The per-element conversion operations in both algorithms [step 5)–step 7) of the first algorithm and step 5) and step 6) of the second] involve only finite field operations. An alternative approach that is also storage-efficient combines finite field operations and dot product operations (a dot product $\underline{a} \cdot \underline{b}$ of two m-bit vectors $\underline{a} = (a_0\, a_1\, a_2 \ldots a_{m-1})$ and $\underline{b} = (b_0\, b_1\, b_2 \ldots b_{m-1})$ is a single bit defined as $a_0\, b_0 \oplus a_1\, b_1 \oplus \ldots \oplus a_{m-1}\, b_{m-1}$). For a polynomial basis, step 5)–step 7) would be replaced with

5. Set $e \leftarrow 1$ (the element one of $GF(2^m)$, represented with respect to B_1).
6. For $i = 0$ to $m - 2$
   6.1 Let $\underline{v}$ be the m-element bit string representing v in B_1, and compute $a_i := \underline{v} \cdot \underline{s}$.
   6.2 If $a_i = 1$, then set $v \leftarrow v + e$.
   6.3 Set $v := vw$.
7. Let $\underline{v}$ be the m-element bit string representing v in B_1, and compute $a_{m-1} := \underline{v} \cdot \underline{s}$.

For a normal basis, step 5) and step 6) would be replaced with

5. For $i = m - 1$ downto 1
   5.1 Let $\underline{v}$ be the m-element bit string representing v in B_1, and compute $a_i := \underline{v} \cdot \underline{s}$.
   5.2 Set $v := v^2$.
6. Let $\underline{v}$ be the m-element bit string representing v in B_1, and compute $a_0 := \underline{v} \cdot \underline{s}$.

In both alternatives, the element z is not needed, so step 3) and step 4) of the algorithm can be omitted and A.7.7 is not needed.

3—If this subroutine is invoked to facilitate division using the Extended Euclidean algorithm of A.4.4 (as described in Note 2 of A.7.6), then in step 1), u and Γ should be computed via A.7.4 rather than A.7.3 [u will simply be the element whose representation with respect to B_1 is (100…0)]. $w = u^{-1}$ should be computed by raising u to the power $2^m - 2$ via A.4.3. (Computing w or the division in A.7.3 via the extended Euclidean algorithm would require one to use basis conversion again, and could thus result in infinitely deep recursion.)

A.9.6 Representation of points

Replace the second bullet under "Coordinate compression" with the following text:

- If q is a power of 2 and the LSB compressed form is employed, then $\bar{y}$ is the rightmost bit of the field element $y\, x^{-1}$ (except when $x = 0$, in which case $\bar{y} := 0$).
- If the SORT compressed form is employed, let y' be the y-coordinate of the inverse point.

(That is, $y' = -y$ if q is odd, and $y' = x + y$ if q is even.) Then $\bar{y}$ is 1 if FE2IP (y) > FE2IP (y'), and 0 otherwise, where FE2IP is defined in 5.5.5.

Replace A.12.2 with the following text:

A.12.2 The Weil pairing

The *Weil pairing* is a function $\langle P, Q \rangle$ of pairs P, Q of points on an elliptic curve E. It can be used to determine whether P and Q are multiples of each other. It will be used in A.12.3.

Let $l > 2$ be prime, and let P and Q be points on E with $lP = lQ = O$. The following procedure computes the Weil pairing.

— Given three finite points (x_0, y_0), (x_1, y_1), (u, v) on E, define the function $f((x_0, y_0), (x_1, y_1), (u, v))$ by

$$\begin{cases} u - x_1 & \text{if } x_0 = x_1 \text{ and } y_0 = -y_1 \\ (3x_1^2 + a)(u - x_1) - 2y_1(v - y_1) & \text{if } x_0 = x_1 \text{ and } y_0 = y_1 \\ (x_0 - x_1)v - (y_0 - y_1)u - (x_0 y_1 - x_1 y_0) & \text{if } x_0 \neq x_1 \end{cases}$$

if E is the curve $y_2 = x_3 + ax + b$ over $GF(p)$, and by

$$\begin{cases} u + x_1 & \text{if } x_0 = x_1 \text{ and } y_0 = x_1 + y_1 \\ x_1^3 + (x_1^2 + y_1)u + x_1 v & \text{if } x_0 = x_1 \text{ and } y_0 = y_1 \\ (x_0 + x_1)v + (y_0 + y_1)u + (x_0 y_1 + x_1 y_0) & \text{if } x_0 \neq x_1 \end{cases}$$

if E is the curve $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^m)$.

Also define $f(O, O, (u, v)) = 1$; $f(O, (x_1, y_1), (u, v)) = u - x_1$ over $GF(p)$ and $u + x_1$ over $GF(2^m)$; and $f((x_0, y_0), O, (u, v)) = u - x_0$ over $GF(p)$ and $u + x_0$ over $GF(2^m)$.

— Given points A, B, C on E, let

$$g(A, B, C) := f(A, B, C) / f(A + B, -A - B, C).$$

— The *Weil function h* is computed via the following algorithm.

Input: A prime $l > 2$; a curve E; finite points D, C on E

Output: The Weil function $h(D, C)$

1) Set $A \leftarrow D$, $B \leftarrow D$, $ha \leftarrow 1$, $hb \leftarrow 1$, $n \leftarrow l$.
2) Set $n \leftarrow \lfloor n/2 \rfloor$.
3) Set $hb \leftarrow hb \times hb \times g(B, B, C)$.
4) Set $B \leftarrow 2B$.
5) If n is odd, then
   5.1) Set $ha \leftarrow ha \times hb \times g(A, B, C)$.
   5.2) Set $A \leftarrow A + B$.
6) If $n > 1$, then go to step 2).
7) Output ha.

— Given points R and S on E, let

$$j(R, S) := h(R, S) / h(S, R).$$

— Given points P and Q on E with $lP = lQ = O$, the Weil pairing $\langle P, Q \rangle$ is computed as follows:

Choose random finite points $T \neq -P$, $U \neq -Q$ on E, and let

$V = P + T, \quad W = Q + U.$

Then

$$<P, Q> = j(T, U)\ j(U, V)\ j(V, W)\ j(W, T).$$

If, in evaluating $<P, Q>$, one encounters $f((x_0, y_0), (x_1, y_1), (u, v)) = 0$, then the calculation fails. In this (unlikely) event, repeat the calculation with newly chosen T and U.

In the case $l = 2$, the Weil pairing is easily computed as follows: $<P, Q>$ equals 1 if $P = Q$ and -1 otherwise.

Define $<P, O> = 1$ for all points P and $<O, Q> = 1$ for all points Q.

Replace the title of A.12.9 with the following:

A.12.9 Decompression of *y*-coordinates (binary case, LSB form)

Change the first paragraph and the input list to the following:

The following algorithm recovers the *y*-coordinate of an elliptic curve point from its LSB compressed form.

Input: A field $GF(2^m)$; an elliptic curve E defined over $GF(2^m)$; the x-coordinate of a point (x, y) on E; the LSB compressed representation $\bar{y}$ of the y-coordinate.

Insert the following subclause after A.12.10:

A.12.11 Decompression of *y*-coordinates (SORT form)

The following algorithm recovers the *y*-coordinate of an elliptic curve point from its SORT compressed form.

Input: A field $GF(q)$; an elliptic curve E defined over $GF(q)$; the x-coordinate of a point (x, y) on E; the SORT compressed representation $\bar{y}$ of the y-coordinate.

Output: The y coordinate of the point.

1) Find field elements z and z' such that (x, z) and (x, z') are elliptic curve points (possibly the same). If none exist, then return an error message and stop.
2) Let $\bar{z}$ be 1 if FE2IP (z) > FE2IP (z'), and 0 otherwise, where FE2IP is defined in 5.5.5.
3) If $z' = \bar{y}$ then $y \leftarrow z$, else $y \leftarrow z'$.
4) Output y.

NOTES

1—This algorithm works for all finite fields defined in this standard. Although it supports odd-characteristic extension fields, the algorithm is defined in this clause rather than A.17 for convenience.

2—The SORT form is only employed for binary fields and odd-characteristic extension fields in the elliptic curve point representations in Clause 5.

3—Finding z and z' requires one to solve the elliptic curve equation at x. Step 1) and step 2) of A.12.8 do this for the prime case and step 1)–step 4) of A.12.9 do so for the binary case (in the notation of A.12.9, the roots are zx and $zx + x$). For odd-characteristic extension fields, a procedure similar to step 1) and step 2) of A.12.8 may be applied, with square roots

implemented via A.17.1.3 rather than A.2.5. Once one element is found, the other can be determined directly; if q is odd, then $z' = -z$, and if q is even, then $z' = x + z$.

A.16.2 An algorithm for validating DL parameters (prime case)

Change the note to the following:

NOTE—A method for verifiably pseudo-random generation of DL parameters is provided in ANSI X9.30:1-1997 [B4], ANSI X9.42-2001 [B8], and FIPS PUB 185186-2 [B56]. (See D.4.1.4 for more information.)

Insert the following subclause after A.16:

A.17 Odd-characteristic extension fields

As defined in A.3.1, a *finite field* (or *Galois field*) is a set with finitely many elements in which the usual algebraic operations (addition, subtraction, multiplication, division by nonzero elements) are possible, and in which the usual algebraic laws (commutative, associative, distributive) hold. The *order* of a finite field is the number of elements it contains. A finite field is identified with the notation $GF(p^m)$ for a prime p and positive integer m. When $m > 1$, field elements are represented as polynomials with coefficients from $GF(p)$. As in 5.3.2.3, if m is composite, the field elements may be represented with a composite basis. In this case, a subfield $GF(p^d)$ of $GF(p^m)$ is chosen where $1 < d < m$ and $d \mid m$. Elements of $GF(p^m)$ are then represented as vectors of elements from $GF(p^d)$, as described by Aoki et al. [AHK99]. Arithmetic is performed exactly as described below, except that arithmetic in $GF(p^m)$ is implemented using operations from $GF(p^d)$. This process of choosing representations for subfields of $GF(p^d)$ may be repeated for each of the divisors of d.

A.17.1 Basic arithmetic in an odd-characteristic extension field

This subclause describes the basic method for arithmetic in fields $GF(p^m)$, for an odd prime p, $m > 1$, of which an Optimal Extension Field (OEF) is a special case. The material in this subclause is described in Bailey and Paar [BP98] (see also [Mih97]).

An extension field $GF(p^m)$ is isomorphic to $GF(p)[t]/(f(t))$, where $f(t)$ is a monic irreducible polynomial of degree m over $GF(p)$. In the following, a residue class will be identified with the polynomial of least degree in this class. The polynomial basis representation of a field element $A(t) \in GF(p^m)$ has the form:

$$A(t) = a_{m-1}t^{m-1} + \ldots + a_1 t + a_0,\ a_i \in GF(p)$$

All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the modular reduction.

A.17.1.1 Addition and subtraction

Addition and subtraction of two field elements is implemented in a straightforward manner by adding or subtracting the coefficients of their polynomial representation and, if necessary, performing a reduction modulo p by subtracting or adding p once from the intermediate result.

A.17.1.2 Extension field multiplication

Field multiplication can be performed in two stages: multiplication and modular reduction. First, perform an ordinary polynomial multiplication of two field elements $A(t)$ and $B(t)$, resulting in an intermediate product $C'(t)$ of degree less than or equal to $2m$-2:

$$C'(t) = A(t) \times B(t) = c'_{2m-2}t^{2m-2} + \ldots + c_1't + c'_0, c'_i \in GF(p)$$

The schoolbook method to calculate the coefficients c^{i}, i = 0, 1, …, $2m$-2, requires m^2 multiplications and $(m-1)^2$ additions in the subfield $GF(p)$, although implementers may achieve better performance using convolution methods.

Second, calculate the residue $C(t) \equiv C'(t) \pmod{f(t)}$, $C(t) \in GF(p^m)$. A.17.3.2 presents an efficient method.

Squaring can be considered a special case of multiplication. The only difference is that the number of coefficient multiplications in the schoolbook method can be reduced to $m(m+1)/2$.

Coefficient multiplications require multiplication in the subfield. A method for fast prime subfield multiplication is found in A.17.3.1. Certain choices for p can result in additional computational savings in the prime subfield multiplication.

A.17.1.3 Square roots in odd-characteristic extension fields

The following algorithm tests whether an element of $GF(p^m)$ is a square.

Input: A field $GF(p^m)$, and an element g

Output: The message “square” or “not a square”.

1) Compute $b = g^{(p^m-1)/2}$.
2) If b is 0 or 1 output “square”; else output “not a square”.

NOTE—b should be 0 only if a is 0, and should otherwise be +1 or -1.

The following algorithm implements a square root in $GF(p^m)$.

Input: A field $GF(p^m)$, and an element g

Output: An element z such that $z^2 = g$, or the message “no square roots exist”:

1) If $g = 0$, then output 0 and stop.
2) Let $k := (p^m + 1)/4$.
3) If k is not an integer, then go to step 6).
4) Let $z := g^k$.
5) If $z^2 = g$ then output z and stop. Otherwise output “no square roots exist” and stop.
6) Choose $y \in GF(p^m)$ at random.
7) If y is a square, then go to step 6).
8) Write $p^m - 1 = k \times 2^e$ with k odd, and let $y := y^k$.
9) Let $A := g^k$, $z := g^{(k+1)/2}$.
10) If $A^{2e-1} \neq 1$, then output “no square roots exist” and stop.
11) If $A = 1$, then go to step 14).
12) Let i be the smallest positive integer such that $A^{2i} = 1$.
13) Let $A := A\,y^{2e-i}$, $z := z\,y^{2e-i-1}$. Go to step 11).
14) Output z and stop.

NOTES

1—Step 3) and step 4) are convenient if $p \equiv 3 \bmod 4$ and m is odd, and they can be skipped in any case.

2—The i of step 12) will always be less than e, assuming that the computations are performed in a valid field (i.e., p is prime, the field polynomial is primitive, etc.). For assurance, implementations may wish to check that $i \leq e$, and output an error message otherwise.

3—Step 11)–step 13) will take at most e iterations, again assuming a valid field. To avoid an infinite loop if the parameters are invalid, implementations may wish to stop explicitly after e iterations.

4—A more efficient square root algorithm is described by Barreto et al. [BKL+02].

A.17.2 Optimal extension fields

Performance improvement can result from the selection of particular finite fields in which the algorithms for extension field arithmetic have an especially efficient implementation.

An *Optimal Extension Field* (OEF) is a finite field $GF(p^m)$ such that:

1) p is a pseudo-Mersenne prime, a positive rational integer of the form $2^n \pm c$, $\log_2 c \leq \lfloor n/2 \rfloor$.
2) An irreducible binomial $f(t) = t^m - \omega$ exists over $GF(p)$. (See Note.)

(To check if the binomial is irreducible in this case, it suffices to check that $\omega^{(p-1)/s} \neq 1 \bmod p$, for all primes s dividing e, where e is the order of ω in $GF(p)^*$. See Lidl and Niederreiter [LN94] for further discussion.)

The characteristic p is often chosen based on the implementation platform. Thus, for a machine with efficient 32-bit arithmetic, one would choose p slightly less than 232. There are two special cases of OEF that yield additional arithmetic advantages, which are referred to as Type I and Type II.

A.17.2.1 Type I optimal extension fields

A T*ype I Optimal Extension Field* has $p = 2^n \pm 1$.

A Type I OEF allows for subfield modular reduction with very low complexity. An OEF may be of both Type I and Type II.

A.17.2.2 Type II optimal extension fields

A *Type II Optimal Extension Field* has an irreducible binomial $f(t) = t^m - 2$.

A Type II OEF allows for a reduction in the complexity of extension field modular reduction because the multiplications by ω in Algorithm A.17.3.2 can be implemented using shifts instead of explicit multiplications. An OEF may be of both Type I and Type II.

A.17.3 Optimal extension fields: Algorithms

A.17.3.1 Subfield multiplication

The following algorithm implements subfield multiplication in an OEF.

Input: A prime $p = 2^n \pm c$, $\log_2 c \leq \lfloor n/2 \rfloor$, integers $0 \leq a, b < p$

Output: The integer $r \equiv ab \pmod{p}$

1) Set $x \leftarrow ab$.
2) Set $q_0 \leftarrow x >> n$.
3) Set $r_0 \leftarrow x - (q_0 << n)$.
4) Set $r \leftarrow r_0$.
5) Set $i \leftarrow 0$.
6) While $q_i > 0$
   6.1) Set $q_{i+1} \leftarrow q_i c >> n$.
   6.2) Set $r_{i+1} \leftarrow q_i c - (q_{i+1} << n)$.
   6.3) Set $i \leftarrow i + 1$.
   6.4) Set $r \leftarrow r + r_i$.
7) While $(r \geq p)$
   7.1) Set $r \leftarrow r - p$.

This algorithm requires a maximum of two iterations of the first while loop, so at most two multiplications by c are required.

If $c = 1$ as in a Type I OEF, this algorithm executes the first while loop only once and multiplication by c is an identity map.

A.17.3.2 Extension field modular reduction

The following algorithm performs extension field modular reduction in the field $GF(p^m)$ modulo an irreducible binomial.

Input: A polynomial $C'(t)$, of degree up to $(2m - 2)$, an integer ω such that $f(t) = t^m - \omega$ is an irreducible binomial over $GF(p)$

Output: A polynomial $C(t) \equiv C'(t) \pmod{f(t)}$, where $C(t)$ is of degree less than m

1) Let $C(t) = c_{m-1} t_{m-1} + \ldots + c_0$ and $C'(t) = c_{2m-2}' t_{2m-2} + \ldots + c_0'$.
2) Set $c_{m-1} \leftarrow c_{m-1}'$.
3) For i from $m - 2$ downto 0, j from $2m - 2$ downto m
   3.1) Set $c_i \leftarrow \omega c_i' + c_i'$.

This algorithm requires a maximum of $(m - 1)$ subfield multiplications by ω. If $\omega = 2^i$ as in a Type II OEF, these multiplications may be implemented as bitwise shifts.

A.17.3.3 Extension field inversion

This algorithm (see Itoh et al. [B81] and Bailey and Paar [BP00]) computes the multiplicative inverse β^{-1} of an element β such that $\beta\beta^{-1} \equiv 1 \in GF(p^m)$. An analogous algorithm for fields $GF(2^m)$ is in A.4.4.

Input: A field $GF(p^m)$ and a nonzero field element β

Output: The reciprocal β^{-1}

1) Set $r \leftarrow (p^m - 1) / (p - 1)$.
2) Set $b \leftarrow \beta^{r-1}$.
3) Set $c \leftarrow b\beta$.
4) Set $\chi \leftarrow c^{-1}$.
5) Set $\beta^{-1} \leftarrow b\chi$.

This algorithm requires an exponentiation to the r-th power and an inversion in $GF(p)$, because $c = b^r = \text{Norm}(\beta) \in GF(p)$, where the function Norm is defined as $\text{Norm}(\alpha) = \alpha \times \alpha^p \times \ldots \times \alpha^{p^{m-1}} = \alpha^r$, where $r = (p^m - 1)/(p - 1)$, $\alpha \in GF(p^m)$.

To quickly perform the exponentiation, observe the following power series representation for r:

$$r = p^{m-1} + p^{m-2} + \ldots + p + 1$$

This corresponds to the p-adic representation $(r - 1) = (11\ldots10)_p$. Exponentiation, then, may be performed by repeatedly multiplying and taking p-th powers, in an analogous manner to the algorithm in A.5.1. As $(r - 1)$ will be fixed for a given field, an addition chain can be used to further reduce the computation required. Using such an addition chain constructed from the p-adic representation of $(r - 1)$ requires:

$(\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1)$ general multiplications + $(\lfloor \log_2(m-1) \rfloor + H_w(m-1))$ exponentiations

where H_w denotes the Hamming weight, that is, the number of 1s in the binary representation of an integer.

Each exponentiation is to a p^i-th power and is thus an iterate of the Frobenius map (i.e., the map $\alpha \rightarrow \alpha^p$). As an OEF has an irreducible binomial as the field polynomial, one has the following algorithm for each exponentiation.

The following algorithm computes the p^i-th power of β, i.e., $\exp(\beta, p^i)$ in an OEF, when $\text{GCD}(p, m) = 1$.

Input: An OEF $GF(p^m)$, $\beta = \sum \beta_j t^j \in GF(p^m)$, an integer i

Output: $\gamma = \exp(\beta, p^i)$

1) For j from $m - 1$ downto 1
   1.1) Set $\gamma_{\pi(j)} \leftarrow \beta_j \omega_{ij}$ where $\pi(j) = jp^i \bmod m$ and $\omega_{ij} = \exp(\omega, \lfloor (jp^i)/m \rfloor)$. (Note that $\omega_{ij} \in GF(p)$.)
2) Set $\gamma_0 \leftarrow \beta_0$.
3) Output $\gamma = \sum \gamma_j t^j$.

As the ω_{ij} values will be fixed for a particular field, they may be precomputed and stored. Thus, this algorithm requires $(m - 1)$ subfield multiplications by fixed elements.

A.17.3.4 Subfield inversion

To compute the subfield inverse required in the algorithm in A.17.3.3, one may use the algorithm in A.2.2.

A.17.3.5 Construction

This algorithm finds an irreducible binomial to construct an Optimal Extension Field, given an approximate subfield order and extension degree.

Input: An integer n, where $2^n \pm c$ will be the characteristic of the field; a positive integer m, the extension degree

Output: ω where $t^m - \omega$ is an irreducible binomial over $GF(p)$.

1) Set $c \leftarrow 1$.
2) While $\log_2 c \leq \lfloor n/2 \rfloor$
   2.1) Set $p \leftarrow 2^n \pm c$.
   2.2) If p is prime and the divisors of m divide $(p - 1)$.
      2.2.1) Set $\omega \leftarrow 2$.
      2.2.2) $\omega < p$
         2.2.3.1) If $t^m - \omega$ is irreducible (see A.17.2), output ω.
         2.2.3.2) Set $\omega \leftarrow \omega + 1$.
   2.3) Set $c \leftarrow c + 2$.

NOTES

1—The performance of this algorithm may be greatly improved by replacing step 2.3) with a sieve step.

2—Under the Extended Riemann Hypothesis, step 2.2.2) will terminate after $O(\log p)$ iterations.

A.17.3.6 Table of type I OEFs for the EC setting

The following table provides Type I OEFs with $2^{160} \leq p^m \leq 2^{256}$.

n	*c*	*m*	*mn*	*ω*
7	-1	27	189	3
8	1	32	256	2
13	-1	13	169	2
13	-1	14	182	17
13	-1	15	195	17
13	-1	18	234	17
16	1	16	256	2
17	-1	10	170	3
17	-1	15	255	3
19	-1	9	171	3
31	-1	6	186	7
31	-1	7	217	7
61	-1	3	183	5

A.17.3.7 Construction in the DL setting

In contrast to their use in the EC setting, an OEF to be used in the DL setting is subject to an additional constraint: The OEF must have a sufficiently large subgroup. An analogous situation exists in the use of prime fields (see D.4.1.4, Note 5).

The problem is more difficult in the OEF case. As noted by Lenstra [Len97], a subgroup is required that is large enough to thwart collision search methods (see D.4.1.4, Note 1), and that is not contained in any $GF(p^n)$ for $n \mid m$, $n < m$. For this purpose, it suffices to verify that the m-th cyclotomic polynomial Φ_m evaluated at p has a sufficiently large prime factor r, which indicates the field has a subgroup of order r subject to the constraints previously mentioned. (The reason that this is sufficient is that, as pointed out in Lenstra [Len97], if $r > m$ divides $\Phi_m(p)$, then r does not divide $\Phi_n(p)$ for any $n \mid m$, $n < m$. As $p^n - 1$ is the product of $\Phi_d(p)$ for $d \mid n$, $d < n$, it follows that r does not divide $p^n - 1$ and hence that the subgroup is not contained in $GF(p^n)$ for any such n.)

The cyclotomic polynomials are given by Riesel [Rie94] as

$$\Phi_m(p) = \Pi_{d|m}(p^d - 1)^{u(m/d)}$$

where μ is the Mobius function defined thus:

$\mu(n) = 1$ if $n = 1$
$\mu(n) = 0$ if n contains a repeated prime factor
$\mu(n) = (-1)^k$ if n is the product of k distinct primes

Thus, for example, to construct an OEF-based DL system with a p represented by 64 bits, one could choose an extension degree of 18 so that the extension field has approximately 2^{1152} elements. This field order exceeds the common choice of a finite field with 2^{1024} elements (see D.4.1.4, Note 1). It remains to determine if an appropriate subgroup exists.

From the above formula, the 18-th cyclotomic polynomial is $\Phi_{18}(p) = p^6 - p^3 + 1$. This expression is an integer of approximately 384 bits, which may be factored by any method, such as the Elliptic Curve Method (see D.4.3.4, Note 3). The largest factor found may be used as the subgroup order r.

In general, $\Phi_m(p)$ will be an integer of $(\varphi(m) \log_2(p))$ bits in length, where φ is the Euler function giving the number of integers coprime to m and less than m. It is thus more efficient to construct an OEF in the DL setting for those extension degrees m with small values of $\varphi(m)$. Even with this restriction, however, there are many fields to choose from. The optimal choice of field parameters will be security level and machine dependent.

A.17.3.8 Table of OEFs for the DL setting

$\log_2 p$	p	m	$\log_2 r$	r	$\log_2 p^m$
58	$2^{58}- 2^{34}+1$	18	161	31383206101329984976881444638533396133679529000057	1044
59	$2^{59}-2^{11}+1$	18	200	188222062239279730297769010625303600800297687351641233-7546233	1062
59	$2^{59}-2^{14}+1$	18	170	28833457183688130947670026450299346310715899103122 31	1062
59	$2^{59}-2^{35}+1$	18	316	16453866784461517299804489674927915183487831440313 5329-309699817283705564219174694625868584754553	1062
60	$2^{60}+2^{33}+1$	18	288	78424037191646640127076692385086452504686240219536 9285-95738042212831773402786078589833 9	1080
64	$2^{64}-2^{32}+1$	18	314	42882872754672261530831141147695815226683494438840 0620-87196651222703068076952299219309748312417	1152
NOTE—The values $\log_2 p$, $\log_2 r$, and $\log_2 p^m$ have been rounded to the nearest integer.					

A.17.4 Elliptic curve algorithms

In general, algorithms for elliptic curves over odd-characteristic extension fields are the same as those given in A.9–A.14 for $GF(p)$ and $GF(2^m)$, and for the most part, the differences are left as an "exercise to the reader." However, some notable points are as follows:

— If $p > 3$, one can follow the Weierstrass equation (cf. A.9) and elliptic curve addition formulae (cf. A.10) for elliptic curves over $GF(p)$, replacing $GF(p)$ directly with $GF(p^m)$. However, if $p = 3$, the Weierstrass equation (see Koblitz [Kob98]) is

$$y^2 = x^3 + ax^2 + b$$

where a and b are elements of $GF(3^m)$ with $a \neq 0$ and $b \neq 0$, and the addition formulae for prime fields in A.10 do not apply. A.17.4.1 gives an algorithm for implementing full addition and subtraction on a curve over $GF(3^m)$ in terms of affine coordinates; algorithms for doubling and for projective coordinates are left to the reader.

— The point compression methods in A.9.6 are not directly applicable.
— The algorithm for computing curve orders in extension fields (A.11.5) can be extended to $GF(p^m)$ essentially by replacing 2 with p.
— If $p > 3$, the algorithm for computing the Weil pairing (A.12.2) is the same as for $GF(p)$. If $p = 3$, the algorithm is the same except that the function $f((x_0, y_0), (x_1, y_1), (u, v))$ is defined as $2a(u\text{-}x_1) – 2y_1(v\text{-}y_1)$ when $x_0 = x_1$ and $y_0 = y_1$.

A.17.4.1 Full addition and subtraction ($p = 3$)

The following algorithm implements a full addition [on a curve over $GF(3^m)$] in terms of affine coordinates. All arithmetic operations are performed in the field.

Input: A field $GF(3^m)$; coefficients a, b for an elliptic curve E: $y^2 = x^3 + ax^2 + b$ over $GF(3^m)$; points $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$ on E

Output: The point $P_2 := P_0 + P_1$

1) If $P_0 = O$, then output $P_2 \leftarrow P_1$ and stop.
2) If $P_1 = O$, then output $P_2 \leftarrow P_0$ and stop.
3) If $x_0 \neq x_1$, then
   3.1) Set $\lambda \leftarrow (y_0 - y_1) / (x_0 - x_1)$, $\lambda \in GF(3^m)^*$.
   3.2) Go to step 7).
4) If $y_0 \neq y_1$, then output $P_2 \leftarrow O$ and stop.
5) If $y_1 = 0$, then output $P_2 \leftarrow O$ and stop.
6) Set $\lambda \leftarrow ax_0 / y_0$.
7) Set $x^2 \leftarrow \lambda^2 - x_0 - x_1 - a$.
8) Set $y^2 \leftarrow (x_1 - x_2)\lambda - y_0$.
9) Output $P_2 \leftarrow (x_2, y_2)$.

This algorithm requires three or four field multiplications and a field inversion.

To subtract the point $P = (x, y)$, one adds the point $-P = (x, -y)$.

A.17.5 Domain parameters and keys

Algorithms for generating and validating DL and EC domain parameters and keys over odd-characteristic extension fields are mostly the same as those given in A.16 for $GF(p)$ and $GF(2^m)$. The differences are as follows:

Generating DL parameters: Same general approach as A.16.3; an outline of the algorithm for OEFs (which is easily generalized to other odd-characteristic extension fields) is given in A.17.3.7.

Validating DL parameters: Same as A.16.2, except that:

— The field polynomial $f(t)$ is also included as an input
— A new step between step 2) and step 3) checks that m is a positive integer, that $f(t)$ is of degree m, and that $f(t)$ is irreducible
— Step 3) checks g with respect to $GF(p^m)$ rather than $GF(p)$ and checks that $r > m$ and that $r \mid \Phi_m(p)$ where Φ_m is the m-th cyclotomic polynomial (see A.17.3.7)
— Step 4) checks that $g^r \equiv 1 \pmod{f(t)}$
— Step 5) and step 6) operate on $p^m - 1$ rather than $p - 1$

Generating DL keys: Same as A.16.5.

Validating DL public keys: Same as A.16.6, except that step 2) of each algorithm checks if $q = p^m$, $p > 3$, $m > 1$, and checks w with respect to $GF(p^m)$ rather than $GF(p)$.

Generating EC parameters: Same general approach as A.16.7.

Validating EC parameters: Same general approach as A.16.8, with some differences in details (e.g., the curve equation for $p = 3$).

Generating EC keys: Same as A.16.9.

Validating EC public keys: Same as A.16.10, except that step 2) and step 3) of each algorithm are replaced with the following:

2) If $q = p^m$, $p > 3$, check that x and y are elements of $GF(p^m)$ and $y^2 = x^3 + ax + b \in GF(p^m)$.
3) If $q = 3^m$, check that x and y are elements of $GF(3^m)$ and that $y^2 = x^3 + ax^2 + b \in GF(3^m)$.

Annex B

(normative)

Conformance

B.2 Conformance requirements

Replace the next to last paragraph with the following text:

The following is a template for a claim of conformance:

> Conforms with IEEE Std 1363-2000 (technique/options) (as amended in this standard) over the region where (constraints on inputs).

Replace Tables B.1 and B.2 with the following tables:

Table B.1—Required subclauses for conformance with primitives

Primitive	Subclauses
DLSVDP-DH	4.2, 6.1, 6.2.1
DLSVDP-DHC	4.2, 6.1, 6.2.2
DLSVDP-MQV	4.2, 6.1, 6.2.3
DLSVDP-MQVC	4.2, 6.1, 6.2.4
DLSP-NR	4.2, 6.1, 6.2.5
DLVP-NR	4.2, 6.1, 6.2.6
DLSP-DSA	4.2, 6.1, 6.2.7
DLVP-DSA	4.2, 6.1, 6.2.8
DLPSP-NR2/PV	4.2, 6.1, 6.2.9
DLSP-NR2	4.2, 6.1, 6.2.10
DLVP-NR2	4.2, 6.1, 6.2.11
DLSP-PV	4.2, 6.1, 6.2.12
DLVP-PV	4.2, 6.1, 6.2.13
ECSVDP-DH	4.2, 7.1, 7.2.1
ECSVDP-DHC	4.2, 7.1, 7.2.2

Table B.1—Required subclauses for conformance with primitives *(continued)*

Primitive	Subclauses
ECSVDP-MQV	4.2, 7.1, 7.2.3
ECSVDP-MQVC	4.2, 7.1, 7.2.4
ECSP-NR	4.2, 7.1, 7.2.5
ECVP-NR	4.2, 7.1, 7.2.6
ECSP-DSA	4.2, 7.1, 7.2.7
ECVP-DSA	4.2, 7.1, 7.2.8
ECPSP-NR2/PV	4.2, 7.1, 7.2.9
ECSP-NR2	4.2, 7.1, 7.2.10
ECVP-NR2	4.2, 7.1, 7.2.11
ECSP-PV	4.2, 7.1, 7.2.12
ECVP-PV	4.2, 7.1, 7.2.13
IFEP-RSA	4.2, 8.1, 8.2.2
IFDP-RSA	4.2, 8.1, 8.2.1, 8.2.3
IFSP-RSA	4.2, 8.1, 8.2.1, 8.2.4
IFVP-RSA	4.2, 8.1, 8.2.5
IFSP-RSA2	4.2, 8.1, 8.2.1, 8.2.6
IFVP-RSA2	4.2, 8.1, 8.2.7
IFSP-RW	4.2, 8.1, 8.2.1, 8.2.8
IFVP-RW	4.2, 8.1, 8.2.9
IFEP-OU	4.2, 8.1, 8.2.10
IFDP-OU	4.2, 8.1, 8.2.11
IFSP-ESIGN	4.2, 8.1, 8.2.12
IFVP-ESIGN	4.2, 8.1, 8.2.13

Table B.2—Required subclauses for conformance with schemes

Scheme	Operation	Subclauses
DL/ECKAS-DH1	Key agreement	4.3, 9.1, 9.2
DL/ECKAS-DH2	Key agreement	4.3, 9.1, 9.3
DL/ECKAS-MQV	Key agreement	4.3, 9.1, 9.4
DL/ECSSA	Signature generation	4.3, 10.1, 10.2.1, 10.2.2
	Signature verification	4.3, 10.1, 10.2.1, 10.2.3
IFSSA	Signature generation	4.3, 10.1, 10.3.1, 10.3.2
	Signature verification	4.3, 10.1, 10.3.1, 10.3.3
DL/ECSSR	Signature generation	4.3, 10.1, 10.3.1, 10.3.2
	Signature verification	4.3, 10.1, 10.3.1, 10.3.3
DL/ECSSR-PV	Signature generation	4.3, 10.1, 10.3.1, 10.5.2
	Signature verification	4.3, 10.1, 10.3.1, 10.5.3
IFSSR	Signature generation	4.3, 10.1, 10.3.1, 10.6.2
	Signature verification	4.3, 10.1, 10.3.1, 10.6.3
IFES	Encryption	4.3, 11.1, 11.2.1, 11.2.2
	Decryption	4.3, 11.1, 11.2.1, 11.2.3
DL/ECIES	Encryption	4.3, 11.1, 11.2.1, 11.3.2
	Decryption	4.3, 11.1, 11.2.1, 11.3.3
IFES-EPOC	Encryption	4.3, 11.1, 11.2.1, 11.4.2
	Decryption	4.3, 11.1, 11.2.1, 11.4.3

Annex C

(informative)

Rationale

Replace C.1.3 with the following text:

C.1.3 How were the decisions made regarding the inclusion of individual schemes?

Three types of schemes are defined in this standard: key agreement, digital signatures, and public-key encryption. In practice, these are the most basic yet important functions in public-key cryptography. Other types of schemes (e.g., identification schemes) may be included in future versions of the standard.

The main purpose for the initial version of the standard (IEEE Std 1363-2000) was to specify basic public-key techniques in a common way. It was recognized that additional techniques remained to be specified, which could later be added to the standard. However, many of those additional techniques required further development before being standardized, whereas the basic techniques in the initial version were relatively more established. It was intended that this amendment would be merged into IEEE Std 1363-2000 during future revisions.

The selection of schemes in this amendment was based on several considerations: an evaluation of the "missing parts" of IEEE Std 1363-2000, which techniques were ready for standardization, and comparison of features and tradeoffs among the proposed techniques. Some reasons for the selection of each individual scheme in IEEE Std 1363-2000 and this amendment can be found in C.3.

Replace C.2.1 with the following text:

C.2.1 Why allow all finite fields for the DL and EC families?

Whereas other standards have limited implementers to prime fields and finite fields of characteristic two [B4], [B11], recent results show that a broader array of options is warranted. Results in Bailey and Paar [BP00] and Lenstra [Len97], for instance, demonstrate the performance advantages that may be gained by allowing implementers to choose finite fields that are well-suited to the underlying hardware and/or admit the use of more efficient algorithms for finite field arithmetic. Such flexibility does not appear to impact security as long as certain guidelines are followed. In fact, Schirokauer et al. [SWD96] show that in the DL setting, prime fields and finite fields of characteristic two are more susceptible to attack than other choices. Finally, new proposals such as one by Schroeppel and Eastlake [SE02] are emerging that support arbitrary finite fields.

Replace C.2.2 with the following text:

C.2.2 Why allow multiple representations for $GF(2^m)$?

Although there is a most commonly used representation for $GF(p)$, there are multiple natural commonly used representations for $GF(2^m)$. There are performance tradeoffs in the use of each representation. When choosing a representation, one needs to consider whether hardware or software performance is more important as well as time complexity, memory complexity, and relevant patents.

Although it would have been possible to standardize one representation and require that applications that want to do computations in a different representation convert from one to the other, the expense of basis conversion may be unacceptable in certain applications. The aim instead was to provide a flexible framework within which applications will decide how to interoperate. Therefore, recommendations on a few popular representations are provided.

C.3 Schemes

C.3.3 For the EC and DL families, why have both DSA and NR signature schemes with appendix?

Replace the last two sentences of the last paragraph with the following:

Should a signature scheme giving message recovery be necessary for the DL or EC system, it can potentially be constructed based on DL/ECSP-NR and DL/ECVP-NR, although new primitives have been introduced for this purpose in the amendment. See the next question in C.3.4.

Replace C.3.4 with the following text (new title):

C.3.4 For the DL and EC families, why have two signature schemes giving message recovery (DL/ECSSR and DL/ECSSR-PV)?

It was anticipated that this standard might include a signature scheme giving message recovery, and ISO/IEC 9796-3:2000 [B79], in draft stage while IEEE Std 1363-2000 was being developed, was mentioned as one possibility. This scheme, called DL/ECSSR here, has been included as it is now an international standard, and is sufficiently secure and efficient. The second scheme, DL/ECSSR-PV, is more bandwidth-efficient than DL/ECSSR, taking advantage of redundancy within the message to minimize the overhead due to the signature. DL/ECSSR-PV is being considered for standardization as part of ISO/IEC 15946-4 [ISO-15946-4].

Replace C.3.5 with the following text (new title):

C.3.5 For the DL and EC families, why was the DL/ECIES encryption scheme selected? (updated; new title)

Several proposals on DL and EC encryption schemes were presented to the 1363 working group during the development of IEEE Std 1363-2000. None of the proposals was considered mature enough to include in IEEE Std 1363-2000, and it was expected that a suitable encryption scheme would be established during the 1363a effort. The DL/ECIES scheme has emerged as a common reference in several other standards efforts, including ANSI X9.63 [B12] and SEC-1 [SEC-1], with accompanying security analysis, and is thus considered appropriate for the amendment. DL/ECIES has two modes: non-DHAES mode, which is compatible with ANSI X9.63 and SEC-1 but does not have the full set of security properties of the DHAES scheme on which it is based (see Abdalla et al. [ABR98]); and DHAES mode, which has the full set. For further discussion, see D.5.3.

Replace C.3.6 with the following text (new title):

C.3.6 For the IF family, why have RSA, RW, and ESIGN signature schemes with the various encoding methods? (updated; new title)

Both RSA and Rabin-Williams (RW) are established signature schemes based on the integer factorization problem. The RSA signature schemes are well understood and widely used. The Rabin-Williams signature verification is generally faster than the RSA signature verification if the public exponent of 2 is used; however, the RSA signature generation has a code-size and speed advantage because there is no need to compute the Jacobi symbol. Both RSA and Rabin-Williams signatures are described in the appendix to ISO/IEC 9796:1991 [B78], in ISO/IEC 9796-2:1997 [ISO-9796-2], and in ISO/IEC 14888-3:1998 [B80].

The ESIGN signature scheme is much faster for signature generation than the RSA and RW schemes, and it is comparable in speed for signature verification. The ESIGN scheme is also described in ISO/IEC 14888-3:1998.

Note that there are two signature and verification primitives (IFSP/IFVP) for RSA, namely, RSA1 and RSA2. RSA1, which specifies just the “raw” RSA operation, has been widely used in various implementations and protocols, such as PKCS #1 ([B126], [PKCS1v2_1]), SET ([B104]), and S/MIME (Dusse et al. [B51] and Dusse et al. [B52]). RSA2 has an extra simple step to adjust the most significant bit of the signature to zero, and hence can save 1 bit (and potentially 1 byte for some key sizes) compared with RSA1. RSA2 is a component in ANSI X9.31-1998 [B7] and the various ISO/IEC standards (although RSA1 is also included in the latest draft revision of ISO/IEC 9796-2 [ISO-9796-2-revision]).

Two new encoding methods for RSA/RW signatures with appendix have been added in the amendment. EMSA3 has been included to align with the various implementations and protocols previously mentioned. Also, a version of RSA/RW signatures for which a security reduction to integer factorization has been given is now offered through the use of the EMSA4 encoding method. (EMSA2, which is also allowed for RSA/RW signatures, is for compatibility with ANSI X9.31.)

Replace C.3.7 with the following text (new title):

C.3.7 For the IF family, why was the IFSSR signature scheme giving message recovery selected? (updated; new title)

A signature scheme giving message recovery based on specified in ISO/IEC 9796:1991 [B78] was included in drafts of IEEE Std 1363-2000 until the last ballot. However, work by Coron et al. [CNS99] and Coppersmith et al. [CHJ99] demonstrated practical chosen-message attacks against this scheme. The working group decided that the attacks warranted exclusion of the scheme from the standard. The working group was not aware of another scheme that would have been ready for the IF family at the time that IEEE Std 1363-2000 was being completed, but anticipated that this amendment might include such a scheme. The PSS scheme (Bellare and Rogaway [B19]) was selected as a basis because a security analysis in the random oracle model was available and because PSS can be defined in terms of the same set of primitives as the IF signature scheme already in IEEE Std 1363-2000 (some modifications were made to address implementation considerations). ISO/IEC JTC1 SC27 (the subcommittee responsible for the ISO/IEC 9796 series of standards) decided to pursue this scheme for a revision of ISO/IEC 9796-2:1997 [ISO-9796-2], with the intent of alignment with this amendment.

Insert the following subclause after C.3.8:

C.3.9 For the IF family, why have two encryption schemes (IFIES and IFIES-EPOC)?

The two schemes, like many others in the standard, offer tradeoffs in terms of security analysis, underlying hard problems and primitives, flexibility, performance, and standardization.

IFIES has a security analysis in the random oracle model based on the RSA primitives and factoring integers of the form $n = pq$. The length of messages it can encrypt is limited by the modulus length (minus any associated padding). IFIES is included in the ANSI X9.44 draft standard [B9] and is being considered as an upgrade for existing use of RSA encryption in practice.

IFIES-EPOC also has a security analysis in the random oracle model but is based on the OU primitives and factoring integers of the form $n = p^2q$. It is more flexible in that the length of messages it can encrypt is not limited. Encryption is typically less efficient with OU compared with RSA, but decryption can be more efficient with appropriate optimizations.

Insert the following subclause after C.3:

C.4 Additional methods

C.4.1 Why were the KDF1 and KDF2 key derivation functions selected?

KDF1 is based on a simple, ad hoc construction that met the requirements of IEEE Std 1363-2000; KDF2 extends KDF1 to provide an arbitrary length output, and it is specified in ANSI X9.63 [B12]. (The design of KDF1 is similar to that of MGF1.) Both are based on an underlying hash function, which is attractive in terms of reducing the number of separate components in an implementation. Note that, as is generally the case for the additional methods in this standard, other key derivation functions are allowed within the framework of the schemes.

It is recognized that KDF1 and KDF2 were not designed to produce keys to encrypt messages of arbitrary length. For direct encryption of a message, symmetric encryption schemes have been introduced (see C.4.4).

C.4.2 Why was the MGF1 mask generation function selected?

MGF1, like KDF1 and KDF2, is a simple, ad hoc construction based on a hash function; it is based on suggestions in the literature for instantiating a mask generation function in a signature scheme. Although other constructions may also appropriate (e.g., a stream cipher, or a block cipher in a certain mode of operation), the hash-function-based design is again attractive in terms of reducing the number of separate components. MGF1, like the key derivation functions, was not designed as a stream cipher; symmetric encryption schemes have been introduced for this purpose (see C.4.4).

C.4.3 Why have SHA-1, RIPEMD-160, SHA-256, SHA-384, and SHA-512?

SHA-1 and RIPEMD-160 are the result of several years of hash function design and are generally considered state-of-the-art for hash functions with 160-bit outputs. SHA-1 has been adopted in a number of standards, including ANSI X9.30:2-1997 [B5], FIPS PUB 186-2 [B56], ISO/IEC 10118-3:1998 [ISO-10118-3], and IEEE Std 1363-2000; RIPEMD-160 is in ISO/IEC 10118-3:1998 and IEEE Std 1363-2000. SHA-1 is perhaps more often recommended, but RIPEMD-160 is also attractive because of its more public design process.

To provide a longer hash output than 160 bits (consistent with higher security levels associated with larger asymmetric key sizes), NIST has introduced three new hash functions, SHA-256, SHA-384, and SHA-512, with the indicated hash output lengths. In anticipation of their adoption in practice, this amendment allows for these hash functions in addition to the previous two that were in IEEE Std 1363-2000. Other hash functions than these are allowed, of course, with the framework of the schemes in this standard.

C.4.4 Why have Triple-DES and AES?

Several of the schemes and methods use an underlying symmetric encryption scheme as part of their operation. There are many choices in this area. This amendment has selected two of the most prominent with a sufficiently large key size: triple-DES, the near-term replacement to the Data Encryption Standard (see ANSI X9.52-1998 [B10] and FIPS PUB 46-3 [FIPS-46-3]), and the Advanced Encryption Standard [FIPS-197], the long-term replacement. In each case, Cipher Block Chaining mode with a common padding rule has been selected because of its wide adoption, but with a null initialization vector to minimize message expansion (see D.5.2.2.6 and D.5.3.2.6 for security rationale). Other modes of operation and algorithms are allowed within the framework of the schemes. In particular, because the recommended methods require that the length of the message in bits is divisible by eight, a different method is needed for implementations that operate on bit strings of arbitrary length.

C.4.5 Why was the MAC1 message authentication code selected?

MAC1 is based on the HMAC function, which has been widely adopted in standards development (see ANSI X9.71-2000 [ANSI-X9.71], FIPS PUB 198 [FIPS-198], and Krawczyk et al. [KBC97]). HMAC is attractive because it can be directly built from an existing hash function and offers a security analysis under reasonable assumptions about the underlying hash function. Other message authentication codes that are more efficient are based on a different class of underlying function (e.g., a symmetric encryption scheme), or they offer a better security analysis might also be adopted, but the working group did not consider any to be sufficiently established for this amendment.

Annex D

(informative)

Security considerations

D.1 Introduction

Change the first paragraph to the following:

This annex addresses security considerations for the cryptographic techniques that are defined in this standard. It is not the intent of this annex to teach everything about security or cover all aspects of security for public-key cryptography. Rather, the goal of this annex is to provide guidelines for implementing the techniques specified in the standard. Moreover, because cryptography is a rapidly changing field, the information provided here is necessarily limited to the published state of the art as of the time the standard was drafted, August 1998, and amended, October 2001. Implementers should therefore review the information against more recent results at the time of implementation; the working group Web page may contain additional relevant information (see http://grouper.ieee.org/groups/1363/index.html).

Insert the following text after the first paragraph:

Some of the techniques given here have security arguments (aka “security proofs”) that claim that certain types of attacks are impossible or impractical. These arguments generally rely on assumptions that themselves are not proved (e.g., that integer factorization is difficult), as well as on restrictions in the type of attack (e.g., the random oracle model). Furthermore, although for some schemes the difficulty of breaking the scheme is argued to be close to the difficulty of the underlying hard problem, for other schemes the relationship is not so “tight.” (For example, the published security bounds given for EME1 are not very tight compared with other schemes; for the most part these distinctions are overlooked in the discussion here.) Finally, as with any research, the security arguments may be subject to improvement or correction. In other words, a claimed “security proof” is not an absolute guarantee of security. Security arguments nevertheless offer some assurance to the implementer, as they imply a more careful analysis by the designer, and faults in a security argument do not necessarily lead to an actual attack. In view of these considerations, the terms “proof,” “security proof,” and “provable security” are avoided in this document, and the more encompassing term “security analysis” is chosen instead.

Insert the following sentence at the end of the last paragraph:

NIST’s recent (draft) recommendations on key management [NIST03a], [NIST03b], [NIST03b] also offer an excellent treatment of the use of methods such as those in this standard to achieve specific security objectives.

D.3.2 Authentication of ownership

Change the last sentence of the third paragraph as follows:

Such methods may be found in Adams and ~~Myers~~ Farrell [~~B1~~AF99], ~~(~~Sections 10.3.3, 10.4, and 13.4.2 of Menezes et al. [B112], and ~~Solo~~ Myers et al. [B141][MLSW00]~~)~~.

D.3.6 Storage of domain parameters and keys

Replace the last paragraph of this subclause with the following text:

When a key is associated with a set of domain parameters, the set of domain parameters may either be stored and distributed explicitly with the key or referenced implicitly. When the set of domain parameters is referenced implicitly, the reference and the set must be protected from unauthorized modification. For instance, suppose a key references a shared set of domain parameters. The reference to the shared set must be unambiguous, and the set must be protected from unauthorized modification. Otherwise, the key may be vulnerable to attack by misuse with incorrect domain parameters. Similar considerations apply if a public component of a private key (e.g., the modulus *n* in an RSA or RW private key) is referenced implicitly. In such a case, the reference and the components must be protected from unauthorized modification or else the private key may be vulnerable to attack by misuse with incorrect public components.

D.4 Family-specific considerations

Replace D.4.1.1 with the following text:

D.4.1.1 Security parameters

The primary security parameters for the DL family are the length in bits of the field order *q* and the length in bits of the subgroup order *r*. A common minimum field order length is 1024 bits, and a common minimum subgroup order length is 160 bits (see ANSI X9.42-2001 [B8]). (Note 1.)

The field type (prime vs. binary vs. odd-characteristic extension field) may affect the security of the schemes. (Notes 2 and 3.)

D.4.1.2 Generation method

Insert the following item after the second item:

— *Odd-characteristic extension field case:* The prime subfield order *p* may be predetermined or generated randomly. The extension degree *m* must be chosen so that the order of the resulting extension field is sufficiently large and a subgroup of adequate size exists.

Replace the last paragraph with the following text:

The domain parameters *q*, *r*, and *g* (and the field representation in the binary and odd-characteristic extension field cases) may be shared among key pairs. The private key *s* should not be shared. (Note 8.)

Replace D.4.1.3 with the following text:

D.4.1.3 Other considerations

The field representation (in the binary and odd-characteristic extension field cases) is not known to affect security, although because the field representation is a domain parameter, it should be protected from unauthorized modification, along with the other domain parameters. (Note 9.)

D.4.1.4 Notes

Replace Notes 1–3 with the following text:

1. *Security parameters:* The security of schemes in the DL family against attacks whose goal is to solve the discrete logarithm problem depends on the difficulties of general-purpose discrete logarithm methods and of collision-search methods. In turn, the difficulty of general-purpose discrete logarithm methods depends on the length in bits of the field order q, and the difficulty of collision-search methods depends on the length in bits of the subgroup order r.

The fastest general-purpose finite field discrete logarithm methods today belong to the index calculus family of algorithms. Some members of this family provide better performance than others depending on the relative sizes of p and m in the field $GF(p^m)$. For example, the Generalized Number Field Sieve (GNFS) (see Gordon [B64], Section 3.6.5 of Menezes et al. [B112], and Schirokauer et al. [SWD96]), which has conjectured asymptotic running time $\exp(((64/9)^{1/3} + o(1)) (\ln q)^{1/3} (\ln \ln q)^{2/3})$, when $m < (\log p)^{1/2 - \varepsilon}$, where $q = p^m$, $o(1)$ denotes a number that goes to zero as q grows, and $\varepsilon > 0$. In particular, large prime fields fall into this category.

For more on estimating the complexity of GNFS for large prime fields, see Note 1 in D.4.3.4; the table given there also applies, except that the memory requirements for the linear algebra are $\log_2 q$ times greater for the discrete logarithm problem than they are for the integer factorization problem. (See also the website http://www.rsasecurity.com/rsalabs/challenges/ on the status of solving the integer factorization problem. The result can be used as an estimate for an appropriate field order in the discrete logarithm system.)

For large fields with relatively small characteristic, such as large binary fields, the function field sieve (see Gordon and McCurley [B66], Section 3.6.5 of Menezes et al. [B112], Schirokauer [B131], and Semaev [B134]) offers the same conjectured asymptotic running time as the GNFS when $m \geq (\log p)^2$. Further improvements for the case of large binary fields bring the asymptotic running time within a constant factor of $\exp(1.587 (\ln q)^{1/3} (\ln \ln q)^{2/3})$. In particular, the method is faster for binary fields than for prime fields.

For odd-characteristic extension fields, one may choose $m^{1/2} < (\log p) < m^2$, as is typical when using an OEF in the DL setting. The best known index calculus algorithms in this case have conjectured an asymptotic running time of $\exp((c + o(1)) (\ln q)^{1/2} (\ln \ln q)^{1/2})$ for some constant c. In particular, the method is faster for binary fields than odd-characteristic extension fields obeying the restriction $m^{1/2} < (\log p) < m^2$.

For all types of field, the Pollard rho method (see Section 3.6.3 of Menezes et al. [B112] and Pollard [B123]), and the related Pollard lambda and other collision-search methods (see, e.g., van Oorschot and Wiener [B122]) can compute discrete logarithms after, on average, $(\pi r/4)^{1/2}$ field multiplications. The memory requirements of such methods are relatively small. (See also http://www.certicom.com/research.html on the status of solving the elliptic curve discrete logarithm problem. The result can be used as an estimate for an appropriate subgroup order in the discrete logarithm system.)

The length of the field order and the length of the subgroup order should be selected so that both the general-purpose methods and the collision-search methods have sufficient running time. Often, the parameters are selected so that the difficulty of both types of method is about the same. It does not have be the same, however. For a variety of reasons, such as availability of hardware, for example, an implementation may choose a larger field or subgroup. As noted, a common minimum field order length is 1024 bits, and a common minimum subgroup order length is 160 bits.

When a set of domain parameters is shared among parties, the size should also take into account the number of key pairs associated with the set, because the total running time for computing k discrete logarithms with the same set of domain parameters is only about $k^{1/2}$ times the running time of computing a single discrete logarithm, provided that more memory is available (van Oorschot and Wiener [B112]). A prudent practice is to pick security parameters such that computing even a single discrete logarithm is considered infeasible.

2. *Field type: prime vs. binary vs. odd-characteristic extension field:* The field type may affect the security of the schemes, because the running time of general-purpose discrete logarithm methods varies based on the relative size of p and m. Further, because the cost of finite field operations may differ among the types of field, the actual running time of collision-search methods may vary by a small factor. Also, as there is only one binary field for each field order length, there are fewer alternatives to choose from, should some binary field be cryptanalyzed, than should a single prime field be cryptanalyzed. Similarly, although there are many OCEFs for a given field order length, there are still more prime fields for a given length than OCEFs.

3. *Prime generation:* In the prime field case, the field order q (i.e., the prime p) should be generated randomly or pseudorandomly, because this provides resistance to special-purpose discrete logarithm methods such as the Special Number Field Sieve, or those given in Gordon [B63].

The prime generation method should have a sufficiently low probability that a nonprime is generated, so that the probability of generating an invalid set of domain parameters is small. A small but nonzero probability of error (say, $< 2^{-100}$) is acceptable because it makes no difference in practice; methods with a small probability of error are generally simpler than those with zero error. See D.6 for more on generating random numbers and A.15 for more on generating primes.

The methods of intentionally constructing a field susceptible to the SNFS found in Gordon [B63] are infeasible for odd-characteristic extension fields because no constructive methods of finding an extension field with a particular subgroup order are known. In addition, the SNFS is unlikely to succeed in the odd-characteristic extension field case due to the difficulty in finding a smoothness bound for large extension fields.

A desired security level can also be provided when the field order has a special form; such a choice requires further security analysis by the implementer.

Computing the field order as a one-way function of a random seed provides a degree of auditing because it makes it difficult to select a prime with some predetermined rare property. It may not entirely prevent the party generating the prime from producing primes with special properties, because the party can try many different seeds, looking for one that yields the desired property, if the property is likely enough to occur. However, it will make it difficult for a party to produce primes that are vulnerable to a special-purpose discrete logarithm method such as the Special Number Field Sieve, provided that the prime is large enough. (A party might seek to introduce such a vulnerability in the interest of determining users' private keys.) It will also make it difficult to mount an attack on DSA in which the opponent picks a particular subgroup order r in order for two messages to have the same signature (see Vaudenay [B146]). (This attack is of concern if the message representative may be larger than the subgroup order r.) ANSI X9.30:1-1997 [B4], ANSI X9.42-2001 [B8], and FIPS PUB 186-2 [B56] give an auditable method of generating primes by incremental search. (As noted previously, as there are no known practical methods for an attacker to find an OCEF with a given subgroup order r, an opponent has no known practical avenue to mount this type of attack in the OCEF case.)

Replace Note 5 with the following text:

5. *Subgroup order:* The subgroup order r may have any value consistent with the other domain parameters (provided it is a primitive divisor of $q - 1$ in the case of binary fields), because the difficulty of the collision-search methods for the discrete logarithm problem depends only on the size of the subgroup order, not on its particular value, as long as it is prime. Selecting a field order that is larger than the size of the hash value

employed in a signature scheme, as is done in ANSI X9.62-1998 [B11], has the benefit of thwarting Vaudenay's attack (Vaudenay [B146]). A small but nonzero probability of error in prime generation or primality testing (say, $< 2^{-100}$) is acceptable because it makes no difference in practice. The subgroup order r should be a primitive divisor of $q - 1$ in the case of binary fields (see A.3.9), because otherwise the subgroup is contained entirely in a proper subfield $GF(2^d)$ of $GF(2^m)$, in which case the opponent need only solve the DL problem in the smaller field $GF(2^d)$.

Likewise in the odd-characteristic extension field case, r should divide $p^m - 1$, but no $p^n - 1$ for $n \mid m$, $n < m$. To ensure this is the case, it suffices to choose r to be a prime factor of the m-th cyclotomic polynomial evaluated at p (see Lenstra [Len97]) (see A.17.3.7 for more details).

Change Note 7 to the following:

7. *(Private keys):* The private key should be generated ~~at random from the range [1, r-1]~~ in an unbiased manner (i.e., randomly or pseudorandomly in the full interval [1, r-1]) because this maximizes the difficulty of recovering the private key by collision-search methods and prevents attacks based on partial information about the private key. A desired level of security can ~~also~~sometimes be provided when the private key is restricted to a large enough subset of the ~~range~~interval (e.g., is shorter than the subgroup order, has low weight, or has some other structure). Such choices require ~~further~~careful security analysis by the implementer. In particular, attacks are known on the various DL/EC signature primitives (see D.5.2.1, Note 3) such that if too much information is available to an opponent about the one-time private key, the signer's private key may be revealed after some number of runs of the primitive. See D.5.2.1, Note 3, for further discussion of these attacks, and for discussion of how to generate integers uniformly at random from some interval. D.5.1.1, Note 1, discusses a similar attack, although one that is less directly linked to choice of the second private key on the DLSVDP-MQV, DLSVDP-MQVC, ECSVDP-MQV, and ECSVDP-MQVC primitives. (See D.6 for more on random number generation.)

Replace D.4.2.1 with the following text:

D.4.2.1 Security parameters

The primary security parameter for the EC family is the length in bits of the subgroup order r. A common minimum subgroup order length is 161 bits (see ANSI X9.62-1998 [B11]). (Note 1.)

The field type (prime vs. binary vs. odd-characteristic extension field) may slightly affect the security of the schemes. (Note 2.)

D.4.2.2 Generation method

Insert the following item after the second item:

— *Odd-characteristic extension field case:* The field order q may have any value that provides a sufficiently large subgroup.

Replace D.4.2.3 with the following text:

D.4.2.3 Other considerations

The field representation (in the binary and odd-characteristic extension field cases) is not known to affect security, although because the field representation is a domain parameter, it should be protected from unauthorized modification, along with the other domain parameters. (Note 10.)

D.4.2.4 Notes

Replace Notes 1 and 2 with the following text:

1. *Security parameters:* The security of schemes in the EC family against many attacks whose goal is to solve the elliptic curve discrete logarithm problem depends on the difficulty of collision-search methods, which are the fastest methods known today for computing discrete logarithms on an arbitrary elliptic curve. The difficulty of collision search methods depends, in turn, on the length in bits of the subgroup order *r*. The Pollard rho (Pollard [B125]) and the related Pollard lambda and other collision-search methods (see, e.g., van Oorschot and Wiener [B122]) can compute elliptic curve discrete logarithms after, on average, $(\pi\, r/4)^{1/2}$ elliptic curve additions. The memory requirements of such methods are relatively small. For prime field anomalous curves, and for curves not satisfying the MOV condition (see A.12.1), there are algorithms known for finding elliptic curve logarithms that are significantly more efficient than the collision methods. For prime field anomalous curves, i.e., elliptic curves over $GF(p)$ having exactly p points, there is an efficient polynomial-time method due to Araki et al. (see Satoh and Araki [B130], Semaev [B134], and Smart [B140]) for solving the elliptic curve discrete logarithm problem. For elliptic curves not satisfying the MOV condition, the elliptic curve discrete logarithm can be solved in subexponential time. Such curves are considered insecure and are, thus, avoided in this standard. Collision-search methods may also be sped-up by a factor of about $(m/e)^{1/2}$ for curves over a binary field $GF(2^m)$ whose coefficients lie in a smaller subfield $GF(2^e)$ (see Gallant et al. [B57] and Wiener and Zuccherato [B148]); this improvement is not considered significant enough to warrant avoidance of such curves, but it may suggest higher key sizes than for general curves. (See also http://www.certicom.com/research.html for more information on the status of solving the elliptic curve discrete logarithm problem.)

The following table provides an estimate for the running time of collisions search methods for different sizes of *r*. The estimate is given in MIPS-Years, where a MIPS-Year is an approximate amount of computation that a machine capable of performing one million arithmetic instructions per second would perform in one year (about 3×10^{13} arithmetic instructions).

Table D.1—Estimated cryptographic strength for EC generator order sizes

Size of the generator order r (Bits)	Pressing time (MIPS-Year)
128	4.0×10^5
172	3×10^{12}
234	3×10^{21}
314	3×10^{55}

There is some variation among published estimates of running time due to the particular definition of a MIPS-Year and to the difficulty of estimating actual processor utilization. (How many arithmetic instructions a modern processor performs in a second when running an actual piece of code depends heavily not only on the clock rate, but also on the processor architecture, the amount and speeds of caches and RAM, and the particular piece of code.) Thus, the estimates given here may differ from others in the literature, although the relative order of growth remains the same. The length of the field order and the length of the subgroup order should be selected so that the collision-search methods have sufficient difficulty. A common minimum subgroup order length is 161 bits.

A result in Gaudry et al. [GHS02] shows that many elliptic curves defined over a binary field with a composite extension degree are vulnerable to attack via a method entirely different from the collision-search approaches. For the field $GF(2^m)$, where $m = ds$, where $d, s > 1$, a large number of curves $E(GF(2^m))$ admit a method of attack with asymptotic time complexity $O(2^{s(2+\varepsilon)})$, where $\varepsilon > 0$. When compared with the collision-search methods, this avenue of attack is asymptotically better when $m \geq 4s$. However, in contrast to the collision-search methods, this approach requires significant overhead, which may make it slower in practice for some parameter choices. More study is needed to fully characterize the break-even point of using the two algorithms. However, as a conservative measure, the draft ANSI X9.63 [B12] and FIPS PUB 186-2 [B56] explicitly exclude the use of these fields. Thus, the use of fields $GF(2^m)$ where $m = ds$ in the EC setting requires further security analysis by the implementer.

The fundamental methods used in Gaudry et al. [GHS02] do not apply to prime fields $GF(p)$ because of the lack of a subfield. The methods also do not apply to binary fields $GF(2^m)$ where m is prime because the unique subfield $GF(2)$ is not suitable. Also, the particular techniques used to obtain the result in Gaudry et al. [GHS02] do not work for odd-characteristic fields $GF(p^m)$.

When a set of domain parameters is shared among parties, the size should also take into account the number of key pairs associated with the set, because the total running time for computing k elliptic curve discrete logarithms with the same set of domain parameters is only about $k^{1/2}$ times the running time of computing a single elliptic curve discrete logarithm, provided that more memory is available (van Oorschot and Wiener [B112]). A prudent practice is to pick security parameters such that computing even a single elliptic curve discrete logarithm is considered infeasible.

2. *Field type: prime vs. binary vs. odd-characteristic extension field:* The field type may slightly affect the security of the schemes, because the cost of finite field operations may differ among the types of field, and the actual running time of collision-search methods may vary by a small factor. Furthermore, as indicated in D.4.1.4, Note 3, there are fewer alternatives to choose from should some binary field or OCEF be cryptanalyzed, than should a single prime field be cryptanalyzed; however, in any case, there are many elliptic curves to choose from, and no method is known that cryptanalyzes all elliptic curves over a given field at the same time. See also the considerations on choice of extension degree in Note 1.

Insert the following paragraph at the end of Note 5:

NIST has published a set of recommended elliptic curves for government use in FIPS 186-2 [B56]. The security parameters for each curve are chosen to correspond to one of five symmetric key sizes (80, 112, 128, 192, and 256 bits), with three curves per key size.

Replace D.4.3.1 with the following text:

D.4.3.1 Security parameter

The primary security parameter for the IF family is the length in bits of the modulus n. A common minimum length is 1024 bits (see ANSI X9.31-1998 [B7]). (Note 1.)

The key type (RSA vs. RW vs. OU vs. ESIGN) may affect the security of the schemes. (Note 2.)

Replace D.4.3.2 with the following text:

D.4.3.2 Generation method

Considerations for generating keys in the IF family include the following:

— The modulus *n* should be sufficiently large. (Note 1.)
— The lengths of the primes *p* and *q* should be approximately half the length of the modulus *n* for RSA and RW keys, and approximately one third the length for OU and ESIGN keys, although other lengths may also provide sufficient security. (Note 3.)
— The primes *p* and *q* should be generated randomly or pseudorandomly with a prime generation method that has a sufficiently low probability of producing a nonprime. If auditing of key generation is required, then the primes should be generated as a one-way function of a random secret seed. (Note 4.)
— For RSA and RW keys, the public exponent *e* may have any value consistent with the modulus and key type; it may be predetermined, or generated randomly or pseudorandomly. For ESIGN keys, the public exponent should be at least $e = 8$. (Note 5.)
— The private exponent d for RSA and RW keys should be derived from the public exponent *e* or generated at random. (Note 6.)
— For OU keys, the values *u* and *v* may have any value consistent with the definition of the key type. The value *u* may be predetermined, or generated randomly or pseudorandomly; the value v_0 from which the value *v* is derived may also be predetermined, or generated randomly or pseudorandomly. (Note 9.)

The public exponent *e* (for RSA, RW, and ESIGN) may be shared among keys. The modulus *n*, the primes *p* and *q*, and the private exponent *d* (for RSA and RW) should not be shared. The values *u* and v_0 for OU keys may be shared among keys. (Note 7.)

D.4.3.4 Notes

Replace Notes 1–3 with the following text:

1. *Modulus size:* The security of schemes in the IF family against attacks whose goal is to recover the private key depends on the difficulty of integer factorization by general-purpose methods, which, in turn, depends on the length in bits of the modulus *n*. For large moduli, the fastest general-purpose factoring method today is the generalized number field sieve (GNFS) (see Buchmann et al. [B32] and Buhler et al. [B34]), which has asymptotic running time $\exp(((64/9)^{1/3} + o(1)) (\ln n)^{1/3} (\ln \ln n)^{2/3})$, where $o(1)$ denotes a number that goes to zero as *n* grows. Previously, the fastest general-purpose method was the multiple polynomial quadratic sieve (MPQS) (see Silverman [B138]), which has running time of about $O(\exp (\ln n \ln \ln n)^{1/2})$. See Section 3.2 of Menezes et al. [B112] and Odlyzko [B121] for more discussion on integer factorization. (See also http://www.rsasecurity.com/rsalabs/challenges/ for more information on the status of solving the integer factorization problem.)

In order to compute the complexity of the GNFS accurately for a particular *n*, one needs to know the precise value of the $o(1)$ term for that *n*. In particular, the complexity of the GNFS is difficult to measure accurately for numbers of cryptographic interest, because they are larger than GNFS can currently factor. The standard practice is to estimate the complexity by extrapolating from running times observed for factoring smaller numbers. Following this, ANSI X9.31-1998 [B7] gives the following estimates for GNFS factorization at various modulus sizes. *Processing time* is the total computational effort, given in MIPS-Years (a MIPS-Year is an approximate amount of computation that a machine capable of performing one million arithmetic instructions per second would perform in one year [about 3×10^{13} arithmetic instructions]); *memory requirements* consider the amount of processor memory for a sieving process, which may be distributed

among many processors, and for linear algebra operations, which would typically be on a single processor; and *storage requirements* is the amount of disk space.

Table D.2—Estimated cryptographic strength for IF modulus sizes

Modulus size	Processing time (MIPS-Year)	Memory requirements (bytes)		Storage requirements (bytes)
		Sieving process	Linear algebra	
512	4.0×10^5	1.3×10^8	2.0×10^{10}	5.0×10^{10}
1024	3×10^{12}	3×10^{11}	1×10^{14}	2×10^{14}
2048	3×10^{21}	8×10^{15}	3×10^{18}	7×10^{18}
4096	3×10^{55}	8×10^{21}	3×10^{24}	7×10^{24}

There is some variation among published estimates of running time due to the particular definition of a MIPS-Year and to the difficulty of estimating actual processor utilization. (How many arithmetic instructions a modern processor performs in a second when running an actual piece of code depends heavily not only on the clock rate, but also on the processor architecture, the amount and speeds of caches and RAM, and the particular piece of code.) Thus, the estimates given here may differ from others in the literature, although the relative order of growth remains the same. The length of the modulus should be selected so that integer factorization methods have sufficient difficulty; as mentioned, a common minimum length is 1024 bits. Very recently, there has been significant discussion on the actual cost of hardware for integer factorization, particularly at the 1024-bit level; Bernstein [Ber01], Lenstra et al. [LST+02], and Shamir and Tromer [ST03] give treatments of this issue. The Shamir–Tromer paper proposes a new hardware implementation that very efficiently parallelizes the processing without the usual cost inflation due to memory requirements, affirming that processing time is the primary metric to consider.

2. *Key type:* The key type may affect the security of the schemes in two respects: first, the set of supported primitives (and possibly schemes as well) depends on the key type; and second, the form of modulus depends on the key type.

The choice between RSA and RW (which is only an issue for the signature schemes) affects security for some encoding methods (EMSA4 and EMSR3) to the extent that a security reduction to integer factorization for a signature scheme with those encoding methods has been given in the RW case but not in the RSA case. For the other encoding methods, the choice is not known to affect the security of the schemes. The relationship with integer factorization is known to vary in general between RSA and RW. For instance, certain attack strategies on signature schemes not specified here (see, e.g., Joye and Quisquater [JQ01]) have been shown to yield the factorization of the modulus in the RW case but not in the RSA case. Also, Boneh and Venkatesan [B30] have recently shown strong evidence that root extraction for a small odd exponent is not equivalent to integer factorization, whereas root extraction for an even exponent of any size is known to be equivalent to integer factorization. Integer factorization is nevertheless the only known approach for forging signatures or recovering messages for the schemes in this standard with the recommended encoding methods, regardless of key type.

Regarding the form of modulus, although it is not known whether moduli of the form $n = p^2q$ (key types OU and ESIGN) are easier to factor than moduli of the form $n = pq$ (key types RSA and RW), some special algorithms to factor moduli of the form p^2q have been studied by several researchers (Peralta [Per97], Peralta and Okamoto [PO96], Pollard [Pol97], Adleman and McCurley [AM95], open problems: C7, O7a, and O7b). Such techniques are variants of the elliptic curve method (ECM) and are several times faster. Recently, Boneh et al. [BDH99] presented an algorithm for factoring moduli of the form $n = p^rq$ with large r using the LLL algorithm (lattice reduction). Their algorithm, however, is only effective for the case where r is large (at least $(\log p)^{1/2}$). If r is constant (or small), the running time of their algorithm is exponential in $\log n$. For large moduli of the form $n = p^2q$ (e.g., at least 1024 bits), their algorithm is less efficient than ECM and GNFS. Therefore, the modulus size for OU and ESIGN keys can be roughly the same as for RSA and RW keys, given current algorithms. (Note that for ESIGN keys, the modulus size must be a multiple of 3 bits in order for the signature and verification primitives to operate correctly.) If a new algorithm were to be found that is more effective for moduli of the form $n = p^2q$ but not for moduli of the form $n = pq$, the modulus size for OU and ESIGN keys might be recommended to be longer than for RSA and RW keys.

3. *Prime size:* For RSA and RW keys, the lengths in bits of the primes p and q should be approximately half the length of the modulus n, because this maximizes the difficulty of certain special-purpose integer factorization methods. Such methods include the Pollard rho method (Pollard [B124]), with asymptotic expected running time $O(p^{1/2})$; the Pollard $p - 1$ method (Pollard [B123]), with running time $O(p')$, where p' is the largest prime factor of $p - 1$; and the $p + 1$ method (Williams [B150]), with running time $O(p')$, where p' is the largest prime factor of $p + 1$. The elliptic curve method (ECM) (Lenstra [B101]) is superior to these; its asymptotic running time is $O(\exp(2\ln p \ln\ln p)^{1/2})$. All of these methods are slower than GNFS (see Note 1 in this subclause) for factoring a large IF modulus with prime factors that are approximately half the length of the modulus. These methods may be effective, however, when one of the prime factors is small.

For OU and ESIGN keys, the lengths in bits of the primes are the same, i.e., approximately one third the length of the modulus, again to maximize the difficulty of special-purpose methods such as those mentioned in Note 1 in this subclause, as well as to meet technical requirements of the schemes based on these keys.

A desired security level can also be provided when the lengths of the primes are not approximately half the length of the modulus, so long as the primes are large enough to resist the special-purpose methods just mentioned. Such a choice requires further security analysis by the implementer. (For some further discussion, see Shamir [B136]; see also Gilbert et al. [B59] for a security analysis.)

Replace Note 5 with the following text:

5. *Public exponent selection (RSA, RW, and ESIGN only):* For RSA and RW keys, the public exponent may have any value consistent with the modulus and key type, because the value of the public exponent is not known to affect the security of the schemes with recommended and correctly implemented encoding methods. IF schemes for these key types with nonrecommended (or incorrectly implemented) encoding methods, or implementations that leak a fraction of the bits of the private exponent or the primes, may be vulnerable to certain attacks when the public exponent is small (see Boneh et al. [B26], Coppersmith et al. [B39], and Hastad [B70]). However, the recommended encoding methods and appropriate protection of the private key (see D.7) prevent these attacks. Typical public exponent values are $e = 2$ for RW keys and $e = 3$ or $2^{16}+1$ for RSA keys. A larger public exponent may nevertheless offer an additional line of defense, as it can mitigate concerns about implementation failures in the encoding methods or in the underlying random number generation for an IF encryption scheme. Moreover, it has been shown that for very small public exponents such as $e = 3$, the RSA problem may not be equivalent to the integer factorization problem (see Boneh and Venkatesan [B30]) (see also Note 2).

For ESIGN keys, the public exponent should be at least $e = 8$. The square degree ($e = 2$) version of ESIGN was proposed in 1985 [OS85] and was broken by Brickell and DeLaurentis in the same year [BD86] (see also Brickell and Odlyzko [BO91]. In other words, they gave an efficient algorithm to solve the approximate square root problem, i.e., the approximate *e*-th root (AER) problem for $e = 2$. They also presented an efficient algorithm to solve the cubic version (AER problem for $e = 3$). In the late 1980s, Girault et al. studied various types of AER problems using lattice reduction [GTV88], [VGT88a], [VGT88b]. (Brickell and DeLaurentis' attack is a special case of their lattice base reduction attack). However, they did not find an efficient solution for $e > 3$. Lattice basis reduction is currently the only effective tool known to solve such AER problems, and no algorithm is known to efficiently solve the AER problem with $e > 3$. (Note that lattice basis reduction is a very powerful tool to solve various problems: for example, almost all knapsack public-key cryptosystems were broken by lattice basis reduction.) To keep a sufficient security margin, it is recommended to choose *e* to be to be at least 8, and more preferably at least 32; typical public exponent values for ESIGN keys are $e = 2^c$ for $5 \le c \le 8$. Note that even if $e = 1024 = 2^{10}$, the computational complexity for exponentiation is fairly comparable (just around three times slower) to that for $e = 8 = 2^3$.

Restricting the allowed set of public exponents system-wide, and ensuring that system components refuse to perform operations with public exponents that do not satisfy the restriction, may provide a measure of protection against protocol attacks that exploit the ability of an opponent to pick an arbitrary public exponent (see, e.g., Chen and Hughes [B37]). However, such attacks are generally better defended against by picking appropriate protocols.

A more common use of a system-wide restriction is picking a specific public exponent system-wide in order to avoid having to store and transmit it with the public key. For example, a system using RW keys may have a system-wide policy that $e = 2$ for all keys, in which case just *n*, not (*n*, *e*), needs to be transmitted and stored for public keys. The system-wide *e* should be protected from unauthorized modification the same way any public key would be.

Replace the title of Note 6 with "Private exponent selection (RSA and RW only)"

Replace Note 7 with the following text:

7. *Key component sharing:* A modulus should not be shared because it is possible, given two public keys with the same modulus and one of the corresponding private keys, to determine the other private key. A prime should not be shared because moduli with one prime factor in common can be factored by taking their GCD. For RSA and RW, a private exponent should not be shared because it is effectively the only private component of the private key. Sharing of any of these quantities is unlikely to occur if primes are generated at random and, for RSA and RW, the private exponent is derived from the public exponent or generated at random. A public exponent for RSA, RW and ESIGN may be shared because it is public and may have any value consistent with the modulus and key type. Similarly, the values *u* and v_0 for OU may be shared.

Change Note 8 as follows:

8. *Private-key representation:* The choice of private-key representation does not affect security against cryptanalytic attack, although security against certain implementation attacks may vary according to the representation. For instance, the Bellcore fault-analysis attack <u>on RSA (or RW) keys</u> (Boneh et al. [B26]) is more feasible if the private key contains the prime factors of the modulus, because a single undetected bit error in an exponentiation modulo one of the primes will compromise the private key. (See D.7 for more on implementation attacks.)

Insert the following note after Note 7 and renumber the existing Note 8 as Note 9:

8. *Other component selection (OU only):* For OU keys, the values u and v may have any value consistent with the definition of the key type. A typical value for u is $u = 2$, which may yield implementation advantages. (Note that $u = 2$ should be appropriate for almost any prime, as the congruence $2^{p-1} \equiv 1 \pmod{p^2}$ is conjectured to hold only for a very small fraction of primes.) A typical value for v_0 (from which v is generated) is $v_0 = u$.

D.5.1.1 Primitives

Insert the following sentence and note at the end of the last paragraph:

Security considerations related to the generation of the second key pair in the -MQV and -MQVC primitives are addressed in the NOTE:

NOTE—For the MQV Secret Value Derivation Primitives (DLSVDP-MQV, DLSVDP-MQVC, ECSVDP-MQV, and ECSVDP-MQVC), Leadbitter and Smart [LS03] have shown in the case of 160-bit r that if an opponent, participating in runs of the primitive, can determine the six most significant bits of the other party's quantity $e = ts + u$, calculated in step 5), and can do this for 20 runs of the primitive, then the opponent can recover the other party's first private key s by solving a discrete logarithm problem that requires about 2^{40} effort. Similar results apply to the least significant bits, and to smaller amounts of bits with greater numbers of primitive invocations and a lower probability of success. It appears that even if one party is known to select the second private key u other than randomly or pseudorandomly in the full interval [1, r-1] (for example, by restricting it to subset of the interval), this knowledge alone is not enough to allow the other party to the primitive to mount the attack. The type of information that would be required to mount this attack is currently only known to be leaked by side-channel analysis, which is out of the scope of this standard.

Replace D.5.1.2 with the following text (new title):

D.5.1.2 Additional methods

D.5.1.2.1 Key derivation functions

The recommended key derivation functions for the key agreement schemes in this standard are KDF1 and KDF2.

A key derivation function for the key agreement schemes in this standard should produce keys that are computationally indistinguishable from randomly generated keys. (See D.6 for more on computational indistinguishability.) KDF1 and KDF2 are considered to have this property, provided that the shared secret input string is sufficiently long and the length of the key derivation parameters is the same for all keys computed from a given shared secret value.

KDF1 and KDF2 have essentially the same design except that KDF2 can produce a key of arbitrary length, whereas the length of the key that KDF1 can produce is limited by the hash function output length. However, although KDF2 can produce an arbitrary length key, it is not recommended for use as a stream cipher for encrypting messages of arbitrary length, as this was not a specific design goal for KDF2. See D.5.3.2.2 for related discussion.

The security of KDF1 and KDF2 depends on the underlying hash function (see D.5.1.2.2). The length of the hash function's intermediate chaining value (for iterative constructions such as in the recommended hash function) governs the security of KDF1 and KDF2 against certain forms of attack. Typical theoretical attacks for distinguishing KDF1 or KDF2 from a random source involve on the order of 2^l operations assuming an l-bit hash function, and these attacks do not necessarily yield any particular key.

D.5.1.2.2 Hash functions

The recommended hash functions for the key agreement schemes in this standard are SHA-1, RIPEMD-160, SHA-256, SHA-384, and SHA-512, the first two of which are options in KDF1 and all of which are options in KDF2.

Although a hash function for a signature scheme in this standard should be collision-resistant (see D.5.2.2.4), this property is not strictly necessary for KDF1 and KDF2 because the shared secret string is unknown. In addition, although the original design goal of a hash function was to support signature schemes, the recommended hash functions are generally considered to be appropriate for a number of applications as well, including key derivation.

Replace D.5.1.6 with the following text:

D.5.1.6 Validation of domain parameters and keys

As discussed in D.3.3, using invalid keys or domain parameters as inputs to a primitive carries with it certain risks. Specifically, for the key agreement schemes, the risks are outlined below. Note that the risks will vary with the particular implementation:

— Failure of the implementation, resulting in an error state
— Reduction in size of the key space for derived keys (see, e.g., Jablon [B83])
— Compromise of information about a private key that is combined with the invalid public key in a secret value derivation primitive (Lim and Lee [B102])

An attack in which an opponent deliberately provides an invalid public key is sometimes referred to as a *small-subgroup attack*. If one does not check whether another party's public key is valid, an opponent may be able to provide an element of small order as the "public key" to confine the shared secret value or to obtain information about the legitimate party's private key. (If both public keys are valid, then the shared secret value ranges over the subgroup of size r generated by the generator g; otherwise, it may be outside the subgroup of size r, possibly in a small set.)

These risks can be mitigated by ensuring the validity of domain parameters and public keys. However, validation does not address certain other risks. A key may be valid and still be insecure (for example, if the random number generation for the key generation was performed improperly, or if the private key is stored insecurely or deliberately disclosed). Therefore, an implementer should consider other ways in which the key may be insecure, and then decide how to appropriately mitigate the risk. FIPS PUB 140-2 [B54] implementation validation and random number generation are typical ways to address some of these concerns (see D.6.1 and D.7).

Cofactor multiplication (i.e., -DHC or -MQVC) is another technique used in defending against invalid public keys, which may require less computation than validating the public key directly. Provided that the associated set of domain parameters is valid and the public key is validated as an element of the appropriate group (i.e., element of the finite field or point on the elliptic curve), a -DHC or -MQVC primitive will operate appropriately on public keys that are in the appropriate group but not in the subgroup of size r; no further validation is necessary.

A situation in which it may be acceptable not to validate a public key is one in which the public key is authenticated (typically by the other party in a key agreement operation) and the private key with which the public key is combined in a secret value derivation primitive has a short cryptoperiod. The authentication prevents an outside opponent from substituting an invalid public key in an attempt to reduce the key space. The short cryptoperiod of the private key mitigates the potential for an adversary to benefit from compromising information about the private key.

The amount of information about the private key that the opponent can possibly compromise by using an invalid public key may be limited if the group has few elements of order less than r (e.g., if $q = 2r + 1$ in the DL case or if the cofactor k is small in the EC case), as described in Lim and Lee [B102]. However, an opponent may still be able to reduce the variability of the shared secret key value by confining it to a small set, such as described in [B83]. Furthermore, especially in the EC case, the public key should be validated as an element of the appropriate group [i.e., a point on the correct elliptic curve, or an element of the correct field $GF(q)$] as specifically defined by the domain parameters. Biehl et al. [BMM00] and Antipa et al. [ABMSV03] describe further potential attacks in the EC case whereby an opponent can compromise the private key of a party who does not validate that the opponent's public key is a point on the correct elliptic curve.

D.5.2 Signature schemes

Replace the second paragraph with the following text:

There are two types of signature schemes: signature schemes with appendix and signature schemes giving message recovery. Signature schemes with appendix require the message to be transmitted in addition to the signature; without knowing the message signed, one cannot verify the signature. Signature schemes giving message recovery produce signatures that contain all or part of the message within them. The verifier does not need to know all of the message in order to verify the signature: if the signature is valid, all or part of the message signed is recovered during the signature verification process. Although signature schemes with appendix may be used to sign messages of practically any length (subject only to the limitations of the encoding method), signature schemes giving message recovery are generally used for messages that are short (although the schemes themselves may well allow larger messages).

A further discussion of desired theoretical properties for signature schemes may be found in Bellare and Rogaway [B19].

NOTE—In general, the use of randomization in signature schemes introduces the possibility that a user (or perhaps the user's device, working against the user) can transmit additional and potentially undetected information separate from the message that is signed, typically by filtering or controlling the randomization in the scheme. All of the schemes except IFSSA with the -RSA or -RW primitives and the EMSA1 or EMSA3 encoding methods (which are deterministic) have this property, which is called a "subliminal channel." The salt value in EMSA4 and EMSR3 is particularly amenable to such purposes because it can be directly specified during signature generation. The one-time public key in DL and EC signature primitives can be used as a subliminal channel (see Simmons [Sim94]). Also, in some situations, the recoverable part of the message may also be considered as providing a subliminal channel because it cannot be recovered without the signer's public key.

To reduce the potential for a subliminal channel, the amount of randomization available for signing a given message should be limited and the verifier (perhaps the user, if concerned that the device is working against the user) should have some way to check this. Deterministic versions of the DL/EC primitives as contemplated in D.5.2.1, Note 2 help somewhat: the same signature is produced each time the same message is signed, thereby removing the channel, but the verifier cannot be sure that the channel has been removed without actually testing the signing implementation (or examining the source code). The situation is the same if the salt value in EMSA4/EMSR3 is derived as a function of the signer's private key and the message representative. In contrast, if the salt value is fixed in EMSR4/EMSR3, the verifier can gain assurance that the channel has been removed simply by observing signatures, without testing the signing implementation directly. However, fixing the salt value reduces the tightness of the security analysis (see D.5.2.2.1, Note 2).

Replace D.5.2.1 with the following text:

D.5.2.1 Primitives

Pre-signature, signature, and verification primitive choices include the following pairs or triples:

— DLSP-NR and DLVP-NR (for DL/ECSSA)
— DLSP-DSA and DLVP-DSA (for DL/ECSSA)
— DLPSP-NR2/PV, DLSP-NR2, and DLVP-NR2 (for DL/ECSSR)
— DLPSP-NR2/PV, DLSP-PV, and DLVP-PV (for DL/ECSSR-PV)
— ECSP-NR and ECVP-NR (for DL/ECSSA)
— ECSP-DSA and ECVP-DSA (for DL/ECSSA)
— ECPSP-NR2/PV, ECSP-NR2, and ECVP-NR2 (for DL/ECSSR)
— ECPSP-NR2/PV, ECSP-PV, and ECVP-PV (for DL/ECSSR-PV)
— IFSP-RSA1 and IFVP-RSA1 (for IFSSA and IFSSR)
— IFSP-RSA2 and IFVP-RSA2 (for IFSSA and IFSSR)
— IFSP-RW and IFVP-RW (for IFSSA and IFSSR)
— IFSP-ESIGN and IFVP-ESIGN (for IFSSA)

The choice among DL, EC, and IF primitives affects security to the extent that the difficulties of the underlying problems differ (see D.4). From a practical perspective, the choice of primitive within a family does not significantly affect security against general attacks, because solving the underlying problem is the best currently known general method for attacking the schemes provided that recommended additional methods are employed (although as discussed in D.4.3, the underlying problem for ESIGN is different than for RSA and RW). The security analysis may vary among the choices; also, certain specific attacks may be relevant for some choices but not for others.

Security considerations related to the generation of one-time key pairs and other values in certain primitives are addressed in Notes 1–3:

NOTES

1—*Precomputation:* The first step in each of DLSP-NR, DLSP-DSA, DLPSP-NR2/PV, and their counterparts in the EC family is to generate a one-time key pair. As this step does not depend on the message being signed, the key pair may be generated in advance of the availability of the message. FIPS PUB 186-2 [B56], Appendix 3.2, gives one precomputation method; additional discussion can be found in Naccache et al. [NMV+95]. (But see Note 3 for caveats about these methods; although the general principles are correct, there is a bias in private key generation that should be avoided in the methods used.) The values r, $\exp(r, e) \bmod n$, and $1 / (e \times \exp(r, e - 1)) \bmod p$ in IFSP-ESIGN may be precomputed by similar means.

2—*Signatures without a random or pseudorandom generator:* If a random or pseudorandom generator is not available for generating the one-time key pair in an implementation of a DL or EC (pre-)signature primitive, an alternative is to apply a key derivation function to the signer's private key and certain key derivation parameters to produce the one-time private key. For instance, the key derivation parameters could equal the message representative, in which case the same value and hence the same signature would be produced each time for the same message. In this case, the signature scheme would be deterministic, and as the one-time key pair would depend explicitly on the message, it could no longer be precomputed in advance of the availability of the message as suggested in Note 1 in this subclause. (It can also help in avoiding subliminal channels as discussed in D.5.2.) This method is analyzed in M'Raïhi et al. [MNP+98] (but see Note 3 for a caveat; see also PKCS #5 v1.5 [PKCS-5-v1_5], Section 5, Note 5 for a similar method in the context of password-based cryptography).

A counter could also be included in the key derivation parameters to avoid the determinism (see 10.4.2 for a situation where this is important). As another alternative, the key derivation parameters could depend on previous messages or message representatives, thereby facilitating precomputation of the one-time key pair before a message is available.

As a clarification to the notes in 6.2.7 and elsewhere, which state that "a new key pair should be generated for every signature," the actual security concern is that different key pairs should be generated for signatures on *different message representatives*. The same "one-time" key pair may be used for multiple signatures on the *same* message representative because in this case the signature operation is merely being repeated. The suggestion above that the one-time key pair be obtained deterministically from the message representative is consistent with this point, as the one-time key will be the same for the same message representative and (with overwhelming probability) different for different message representatives. (Note that the same message may produce different message representatives if the encoding method is randomized, hence, the suggested dependence of the one-time key pair on the message representative rather than the message itself.)

(Similar remarks to the foregoing apply to the value *r* in IFSP-ESIGN and the salt value in EMSA4/EMSR3.)

3—*One-time private key generation:* The one-time private key *u* in the DL and EC primitives should be selected in an unbiased manner (i.e., randomly or pseudorandomly in the full interval [1, *r*-1]), as otherwise information about the signer's private key *s* may be leaked by signature primitive. In particular, following earlier work by Howgrave-Graham and Smart [HS01], Nguyen [Ngu01] and Shparlinski and Nguyen [NS02] have shown that if an opponent can determine the three least significant bits of the one-time private key in each of about 100 DLSP-DSA signatures, the opponent can solve for the signer's private key. They also give attacks for the case that other bits of the one-time private key are known. A similar attack applies to ECSP-DSA (because the signature equation is essentially the same after the one-time key pair is generated) and a related (though more complex) attack applies to the -NR (and hence -PV) primitives (see Nguyen and Shparlinski [NS03] and El Mahassni et al. [ENS01]).

Bleichenbacher [Ble01] has given an even more effective attack that can recover the signer's private key given only a slight bias in the distribution of one-time private keys, without actually knowing any bits of the one-time private keys. This attack is particularly significant in that methods suggested for generating one-time private keys in ANSI X9.30:1-1997 [B4] and FIPS PUB 186-2 [B56] as well as in the Naccache et al. and M'Raïhi et al. proposals above have exactly the kind of bias that Bleichenbacher's attack exploits, as they directly use a hash function to generate a value in an interval slightly larger than the subgroup order *r*. Specifically, for a 160-bit value *r*, a one-time private key is produced by pseudorandomly generating a 160-bit string, converting it to an integer in the interval $[0, 2^{160}-1]$, and then reducing the integer modulo *r*. As a result, the private key is biased toward the lower end of the interval [1, *r*-1], which is sufficient to enable the attack.

Bleichenbacher's attack is currently infeasible as long as only on the order of 1 000 000 signatures are generated with a given private key. Hence, limiting the number of signatures is an interim solution to prevent the attack. To avoid the potential for further attacks, it is preferable to produce the one-time private keys in a different way. Two examples are the following:

— *Select from a larger interval before reducing*. Generate a 2*l*-bit string where $l = \lceil \log_2 r \rceil$, convert it to an integer in the interval $[0, 2^{2l}-1]$, and then reduce it modulo *r*. Assuming the string is uniformly distributed, the private key will be nearly uniformly distributed. This method is attractive because it has a fixed running time.
— *Reselect rather than reducing*. Generate an *l*-bit string where $l = \lceil \log_2 r \rceil$, convert it to an integer in the interval $[0, 2^{l}-1]$, and if the integer is not between 1 and $r - 1$, select another one. Assuming the string is uniformly distributed, the private key will also be uniformly distributed. This method has a variable running time.

Variations of these methods may also provide acceptable security, but methods with greater bias than the above ones should be carefully analyzed to ensure that the security is indeed acceptable. For instance, in the first method, the length of the string does not have to be exactly 2*l*-bits, just sufficiently longer than *l*-bits to ensure a nearly uniform distribution after reduction. Specific countermeasures are given in FIPS PUB 186-2, Change Notice 1, and expected to be added to ANSI X9.30:1.

Replace D.5.2.2 with the following text (new title):

D.5.2.2 Additional methods

D.5.2.2.1 Message-encoding methods for signatures with appendix

The recommended encoding methods for signatures with appendix are EMSA1 (for DL/ECSSA) and EMSA2, EMSA3, EMSA4, and EMSA5 (for IFSSA; the recommendation varies depending on the underlying primitives).

An encoding method for a signature scheme with appendix in this standard should have the following properties, stated informally:

— It should be difficult to find a message with a given message representative (the *one-way* property).
— It should be difficult to find two messages with the same representative (*collision resistance*).
— The encoding method should have minimal mathematical structure that could interact with the selected signature primitive (e.g., if the signature primitive is multiplicative, the encoding method should not be).

EMSA1 is considered to have these properties for DL/ECSSA, and EMSA2, EMSA3, EMSA4, and EMSA5 are considered to have these properties for IFSSA (with respect to the appropriate primitives).

The security of all four encoding methods depends on the underlying hash function (see D.5.2.2.3). The security of EMSA4 depends on the underlying mask generation function (see D.5.2.2.4). Other considerations include

— For EMSA1, the relationship between the maximum length of the message representative and the output length of the underlying hash function (Note 1)
— For EMSA4, the length and randomness of the salt value (Note 2)
— Whether the message representative identifies the underlying hash function (Note 3)

NOTES

1—*Message representative length for EMSA1:* For EMSA1, if the maximum length l of the message representative is less than the output length of the hash function, then EMSA1 will simply truncate the hash function output. Thus, for DL/ECSSA, if the length of the subgroup order r (and, hence, the maximum length of the message representative) is less than the length of the hash function output (e.g., 160 bits for SHA-1), the security of the encoding method will be limited by 2^l and $2^{l/2}$, rather than 2^{160} and 2^{80}, respectively. In addition, for DLSP-DSA and ECSP-DSA, if the length of r is not greater than the length of the hash function output, then an opponent may pick r in such a way as to cause two different messages to have the same signature (see Vaudenay [B146]). This can be prevented by increasing the length of r beyond the length of the hash function output, or by auditing domain parameter generation (see Note 3 in D.4.1.4 and Note 5 in D.4.2.4). It is customary for the length of r and the length of the hash function output to be approximately equal, so the message representative input to the signature primitive spans most or all of the range between 0 and $r - 1$. If the length of r is greater than the length of the hash value, then this condition will no longer hold, but this is not known to result in any security risk.

2—*Salt length and randomness for EMSA4 and EMSR3:* In the security analysis for the schemes, the ratio of complexity between forging signatures and inverting the underlying signature primitive depends on the length and randomness of the salt value. Roughly speaking, with a random salt that is as long as the hash output, the ratio approaches 1; with a fixed salt or with no salt, the ratio approaches $1/Q$, where Q is the number of "queries" (chosen messages) involved in the attack. (See Coron [Cor00], [Cor02] for a precise analysis in the context of the original PSS and Full-Domain-Hash (FDH) schemes.) Thus, even with no salt, the analysis suggests that signature forgery is about $1/Q$ times as difficult as inverting the verification primitive (this is a lower bound, so it may actually be more difficult). See D.6 for recommendations on random number generation.

3—*Hash function identification:* An encoding method may identify the hash function (and other options) within the message representative in order to prevent an opponent from tricking a verifier into operating with a different hash function than the signer intended, perhaps a weak hash function, in an attempt to forge a signature. The encoding methods vary with respect to this property (see Kaliski [Kal02] for further analysis):

— EMSA1 does not identify the hash function, but in practice is typically combined with a single hash function, SHA-1, which amounts to a key usage restriction and removes any risk of ambiguity.
— EMSA2 and EMSA3 both identify the hash function and offer protection against hash function substitution.
— EMSA4 includes the option of identifying the hash function, which protects against reuse of existing signatures with a different hash function. Even if the hash function is not identified, however, EMSA4 protects against hash function substitution (both for existing and new signatures) under reasonable assumptions. In particular, because the mask generation function is based on the hash function, EMSA4 protects against hash function substitution by making it difficult to obtain new message representatives with appropriate structure for any "natural" hash function. This is subject to the condition that a minimum amount of padding (e.g., 80 bits) is included

in the message representative, so that even if the hash function underlying the mask generation function turns out to be weak, the opponent still will not be able to find a signature that yields the correct padding. In EMSA4, the minimum will typically be met as the hash output length is significantly shorter than the message representative length.

The foregoing is based on the assumption that the hash function is a "natural" one, selected independently of the signer's public key; it is certainly possible to "cook" a hash function so that the associated mask generation function correctly matches any amount of padding. But now the opponent would need to coerce the verifier into using this "cooked" hash function, a much less plausible scenario than tricking the verifier into using a "natural" hash function that happens to be weak.

One motivation for not identifying the hash function in EMSA4 is that the security analysis is affected by the number of bits of the message representative having fixed values. The fewer, the better, hence the attempt in EMSA4 to minimize the number of fixed values (compared with, say, the original ISO/IEC 9796-2:1997 format [ISO-9796-2]).

Protection against hash function substitution may also be accomplished by other means. For instance, only a small number of well-trusted hash functions might be accepted in a domain. Alternatively, key usage restrictions associated with the signer's key (see D.3.5) could indicate the acceptable hash functions for that key. Hash function identification in signature schemes is an area of current study in ISO/IEC JTC1 SC27.

D.5.2.2.2 Message-encoding methods for signatures giving message recovery

The recommended encoding methods for signatures giving message recovery are EMSR1 (for DL/ECSSR), EMSR2 (for DL/ECSSR-PV), and EMSR3 (for IFSSR).

An encoding method for a signature scheme giving message recovery in this standard should have the following properties:

— It should produce message representatives from which a message (or recoverable message part) can be easily and uniquely recovered (*invertibility*).
— It should produce message representatives that have some verifiable structure, so that it is difficult, without knowing the private key, to forge a signature from which a valid representative can be recovered (*redundancy*).
— The encoding method should have minimal mathematical structure that could interact with the selected signature primitive.

EMSR1 is considered to have these properties for DL/ECSSR, and EMSR3 is considered to have them for IFSSR. EMSR2 is considered to have these properties for DL/ECSSR-PV provided the agreed redundancy criteria and the parameter *padLen* give a sufficient total amount of redundancy. (In general, the role of the encoding method in DL/ECSSR-PV is different than the role of the encoding methods in the other signature schemes, and the required properties are accordingly different. For instance, the third property is less important for EMSR2, because in DL/ECSSR-PV, the output of the encoding method is processed with a hash function before it is input to the signature primitive.)

The security of EMSR1 and EMSR3 depends on the underlying hash function (see D.5.2.2.3). The security of EMSR3 also depends on the underlying mask generation function (see D.5.2.2.4). Other considerations include

— For EMSR1, the redundancy lengths (Note 1)
— For EMSR2, the method of combining the pre-signature with the padded message part (Note 2), the amount of redundancy in the message representative (Note 3), and the encoding of the message parts (Note 4)
— For EMSR3, the length and randomness of the salt value (see D.5.2.2.1 Note 2)
— Whether the message representative identifies the underlying hash function (Note 5)

NOTES

1—*Redundancy lengths for EMSR1:* To maintain the security level of the underlying hash function in general, the long redundancy length for EMSR1 should equal the output length of the hash function plus the length of any hash function identifier. As hash function collisions are not a concern in the case of total message recovery, the short redundancy length is recommended to be at least half the output length of the hash function if hash function identification is not desired. However, if hash function identification is desired, it should also equal the output length of the hash function plus the length of any hash function identifier.

2—*Method of combining the pre-signature with the padded message part for EMSR2:* If a symmetric encryption scheme is used to combine the pre-signature with the padded message part in EMSR2, then the security depends on the key size of the symmetric encryption scheme. For example if a symmetric encryption scheme with a 56-bit key size is used and (C, d) is a valid signature on a message with nonrecoverable message part M_2, then the probability that (C, d') will verify for a random d' and any given M_2' will be roughly 2^{-56}. This probability of forgery may be too high. Similarly, if the key derivation function has a high probability of producing collisions when evaluated with a given padded message part and random pre-signatures, then security may be compromised. These concerns highlight the importance of selecting a sufficient key size and a good key derivation function such as the recommended key derivation function KDF2.

The primary security attribute of the "combining method" for EMSR2 seems to be combining both inputs (pre-signature and padded message part) in a "reversible" way that is highly dependent on both inputs. The additional attributes of confidentiality protection required by symmetric encryption schemes and key derivation functions when used for encryption are not strictly necessary for EMSR2. Accordingly, a stream cipher based on a key derivation function may offer adequate security for EMSR2, even if the key derivation is not sufficiently strong for encryption in general. Nevertheless, the heuristic of "ideal randomness" in a strong symmetric encryption scheme with sufficient key size might offer better security than a stream cipher based on a key derivation function. Further details may be found in Brown and Johnson [BJ00].

3—*Amount of redundancy in the message representative for EMSR2:* The total amount of redundancy in the message representative for EMSR2 is at least $8 \times padLen + agreed$ bits, where *agreed* is the amount of redundancy in bits within the recoverable message part M_1 ensured by the agreed redundancy criteria for the acceptance of a message by both parties. (Redundancy in the nonrecoverable message part M_2 does not affect the value of *agreed*.) Implementations should ensure that the total redundancy is at least at an acceptable level, which is generally half the length of the subgroup order r, or half the length of the hash function output. Implementations may reduce the amount of padding *padLen* to shorten the signature, provided the *agreed* value is high enough to make the total redundancy acceptable. The amount of padding in a symmetric encryption scheme underlying EMSR2 may also be considered in determining the total amount of redundancy.

4—*Prefixes and redundancy for EMSR2:* Suppose the redundancy criteria are such that a message representative C and one of its prefixes C' both yield acceptable recoverable message parts for a given pre-signature I. Then it may be possible in DL/ECSSR-PV2 to forge a new signature involving C' and a new M_2' from a genuine signature involving C and M_2 by writing $C = C' \| T$ and $M_2' = T \| M_2$. (Both cases will yield the same hash value H and hence the same pre-signature I for a given public key.)

There are number of straightforward ways to address this concern. For instance, the redundancy criteria might specify that the recoverable message part has a fixed length, or that it begins with a fixed-length representation of its length. Another example of redundancy criteria that typically disallows such prefixes is the use of a DER encoding of an ASN.1 type. For the recommended options in EMSR2, these criteria ensure that the set of message representatives that yield acceptable recoverable message parts for a given pre-signature I is prefix-free (nonrecommended options require further security analysis by the implementer).

As alternative approaches, rather than adding to the necessary redundancy criteria for the message representative, one might address the concern by specifying that the nonrecoverable message part has a fixed length (perhaps empty), or that it ends with a fixed-length representation of its length.

5—*Hash function identification:* Continuing the discussion in D.5.2.2.1, Note 3, the encoding methods vary with respect to hash function identification as follows:

- EMSR1 includes the option of identifying the hash function, but the hash function identifier does not offer the full claimed protection against hash function substitution. The option is thus included for compatibility with ISO/IEC 9796-3:2000 only.
- EMSR2 does not identify the hash function and does not offer protection against hash function substitution.

— EMSR3 includes the option of identifying the hash function, which protects against reuse of existing signatures with a different hash function. Even if the hash function is not identified, however, protection against hash function substitution (both for existing and new signatures) may be obtained under similar conditions as with EMSA4. If there is a risk of ambiguity in the hash function, the recoverable part of the message should be somewhat shorter than the maximum length allowed so that there is sufficient padding (e.g., 80 bits). (This recommendation applies whether the hash function is identified or not.)

D.5.2.2.3 Key derivation functions

The only recommended key derivation function for the signature schemes in this standard is KDF2, which is an option in EMSR2.

As noted in D.5.1.2, a key derivation function for a key agreement scheme in this standard should produce keys that are computationally indistinguishable from randomly generated keys, and KDF2 is considered to have this property provided that the shared secret input string is sufficiently long. However, as stated in D.5.2.2.2 Note 2, this property is not strictly necessary for EMSR2.

D.5.2.2.4 Hash functions

The recommended hash functions for the signature schemes in this standard are SHA-1, RIPEMD-160, SHA-256, SHA-384, and SHA-512, which are options in DL/ECSSR-PV and all of the encoding methods for signatures except EMSR2 (which does not invoke a hash function), as well as in KDF2 and MGF1.

A hash function for a signature scheme in this standard should generally be collision-resistant; the recommended hash functions are considered to have this property (with respect to a level of difficulty that depends on the hash output length).

Typical attacks for finding a message with a given representative based on hash function inversion involve on the order of $2l$ operations assuming an l-bit hash function; attacks for finding two messages with the same representative based on hash function collision involve $2^{l/2}$ operations. However, the actual complexity and applicability of an attack is sometimes different. For instance, in the case of total message recovery for EMSR1 and EMSR3, collision resistance seems not to be required because even if two recoverable message parts have the same hash value, they will have different message representatives, so the signature on one recoverable message part cannot be used to recover the other. A collision-resistant hash function is nevertheless recommended as it is a prudent security choice; a different choice requires further security analysis by the implementer.

Note that collision resistance does also not seem to be strictly necessary in the original PSS and PSS-R proposals (Bellare and Rogaway [B19]), because the message was hashed together with a random salt value. As the message is hashed directly in EMSA4 and EMSR3, however, the collision resistance is again required.

In addition to processing the message directly in EMSA4 and EMSR3, the hash function also instantiates a random oracle. As further discussed with respect to mask generation functions in D.5.2.2.5, no formal assumptions are known for hash functions that facilitate security analysis outside the random oracle model for these schemes. However, a collision-resistant hash function is a reasonable design approach for ensuring that specific attacks are implausible. For similar reasons, a collision-resistant (and robustly designed) hash function is recommended for constructing a mask generation function.

Hash function properties use with a key derivation function are discussed in D.5.1.2.2.

D.5.2.2.5 Mask generation functions

The only recommended mask generation function for the signature schemes in this standard is MGF1, which is an option in EMSA4 and EMSR3.

Reflecting the "random oracle" that it instantiates, the mask generation function should be pseudorandom: Given one part of the output but not the input, it should be infeasible to predict another part of the output. MGF1 is considered to have this property provided the octet string input is sufficiently long. However, it should be noted that no formal assumptions are known for mask generation functions that facilitate security analysis in the "standard model" (i.e., without random oracle assumptions) for the schemes based on EMSA4 and EMSR3. Moreover, security analysis in the random oracle model only addresses "generic" or "black-box" attacks, and it does not assess whether attacks may exist for specific instantiations with an actual mask generation (or hash) function (see also Canetti et al. [CGH98] for discussion). Nevertheless, pseudorandomness is a reasonable design approach to ensure that specific attacks are implausible.

The security of MGF1 depends on the underlying hash function (see D.5.2.2.4). To defend against hash function substitution (D.5.2.2.2, Note 5), the underlying hash function for MGF1 should be the same as the selected hash function for the encoding method that invokes it.

D.5.2.2.6 Symmetric encryption schemes

The recommended symmetric encryption scheme for the signature schemes in this standard are Triple-DES-CBC-IV0 and AES-CBC-IV0, which are options in EMSR2.

As indicated in D.5.2.2.2, Note 2, not all of the attributes of a symmetric encryption scheme are strictly necessary for EMSR2; the goal is that the key and the message should be combined reversibly in a way that is highly dependent on both inputs. A sufficiently large key size, say, 80 bits or more, is recommended to minimize the probability that a random signature will verify. Triple-DES-CBC-IV0, with an effective key size (against brute-force search) of at least 112 bits, and AES-CBC-IV0, with at least 128 bits, are considered to provide the necessary properties for EMSR2 (although these are clearly not the only symmetric encryption schemes that may provide adequate security in this context).

D.5.3 Encryption schemes

Replace the first paragraph with the following text:

Encryption schemes are generally used to provide confidentiality of data. A theoretical definition of "semantic security" (or, equivalently, "indistinguishability" or "polynomial security") is commonly used as the basis for the meaning of "confidentiality" (see Goldwasser and Micali [B61], Section 8.7 of Menezes et al. [B112], Micali et al. [B114], and Yao [B151]). Semantic security provides that it should be computationally infeasible to recover any information about the plaintext (except its length) from the ciphertext without knowing the private key. This implies, in particular, that it should also be infeasible to find out whether two ciphertexts correspond to the same, or in some way related, plaintexts, or whether a given ciphertext is an encryption of a given plaintext. The encryption schemes in this standard are believed to provide semantic security against an adaptive chosen ciphertext attack (an attack in which the opponent requests decryptions of specially chosen ciphertexts in order to be able to decrypt other ciphertexts) when the appropriate scheme options are selected.

Insert the following paragraph after the first paragraph:

The encryption schemes in this standard are constructed in two different ways. DL/ECIES is built from a secret value derivation primitive, a key derivation function, and a message authentication code; IFES and IFES-EPOC are built from a pair of encryption and decryption primitives, and an encoding method for encryption. All of the encryption schemes support encoding parameters. DL/ECIES also supports key derivation parameters. Accordingly, some of the discussion below applies only to DL/ECIES, and some only to IFES and IFES-EPOC.

Insert the following notes:

NOTES

1—DL/ECIES differs from the original Abdalla-Bellare-Rogaway DHAES submission [ABR98] in that it offers the choice of whether or not to include a representation of the sender's public key *v* as an input to the key derivation function (DHAES mode vs. non-DHAES mode). If a representation of the public key is not included, the scheme is *malleable* in the formal sense that an opponent can replace a ciphertext (*V, C, T*) with a different ciphertext (*V′, C, T*) that yields the same message *M*, for any *V′* such that SVDP (*s, v*) = SVDP (*s, v′*), where SVDP is the selected secret value derivation primitive and *v* and *v′* are the public keys corresponding to the representations *V* and *V′*. (If a scheme is malleable, then formally it is not secure against an adaptively chosen ciphertext attack.) For instance, when SVDP includes cofactor multiplication, a component outside the subgroup can be added to *v* without affecting the output of SVDP (except for elliptic curve groups with a prime number of points, which have no components outside the subgroup). In addition, for elliptic curve groups, *v* can be inverted without affecting the output of SVDP, regardless of whether SVDP includes cofactor multiplication. Shoup has also observed in his write-up assessing encryption schemes for the ISO 18033 project (see Shoup [Sho01b], Section 15.6.1) that leaving out the sender's public key also appears to weaken the security analysis of the scheme.

More generally, in non-DHAES mode, the *representation* of the sender's public key in the ciphertext can be changed (e.g., compressed vs. uncompressed points) without affecting the output of SVDP. This raises primarily theoretical concerns because the resulting message is the same. Indeed, every encryption scheme might be malleable in this theoretical sense because of non-unique representations of the ciphertext commonly used in transmission (as is the case with BER encoding). If assurance of the exact representation of a ciphertext is essential, other authentication methods, especially those using a private key of the sender, should be considered.

Non-DHAES mode is included for compatibility with related standards efforts. The original motivation for leaving out the sender's public key in those standards efforts was to reduce the size of the input to KDF and thereby reduce processing time, although one can argue that the processing time for KDF is likely to be small in general compared with the time to generate a key pair and to apply the secret value derivation primitive. However, where it is practical to do so, the sender should include a representation of the public key as an input to the key derivation function for increased assurance. DHAES mode does this automatically. In non-DHAES mode, this can be done by "defining" the key derivation parameters so that they include the public key. In either case, if the sender's public key is included, the sender and recipient both have to be able to process whatever representation is selected for the public key when it is input to the KDF (e.g., compressed vs. uncompressed points, as above).

As Shoup has also shown ([Sho01b], Section 15.6.4), the message length in non-DHAES mode with the stream cipher option should be fixed for a given public key, because otherwise the encryption scheme is malleable due to the incorrect ordering of the encryption and MAC keys (K_1 and K_2 in the scheme). In DHAES mode, the ordering is corrected so this is not an issue.

2—In general, a recipient's public key should be used for only one set of scheme options to prevent undesirable interactions between different options. As an example of such interactions, if a recipient's public key in DL/ECIES may be used either in combination with a stream cipher based on a key derivation function and in combination with a symmetric encryption scheme, it may be possible for an opponent to transform a ciphertext generated with the symmetric encryption scheme into that which appears to have been generated with the stream cipher and that which corresponds to a different message. One way to avoid this interaction if both options are supported is to include an indication of which option is selected in the key derivation parameters.

3—The specific choice of primitives for converting elliptic curve points to and from octet strings is not known to affect security. However, to avoid any potential for attacks based on "collisions" between different primitives (i.e., where the same octet string corresponds to different points in different representations), it is beneficial if the sets of octet strings for different representations are nonoverlapping (except perhaps at the point at infinity). This is the case for the various primitives specified in 5.5.6.

Replace D.5.3.1 with the following text:

D.5.3.1 Primitives

Secret value derivation primitive choices for DL/ECIES include the following:

— DLSVDP-DH
— DLSVDP-DHC
— ECSVDP-DH
— ECSVDP-DHC

Encryption and decryption primitive choices for IFES and IFES-EPOC include the following pairs:

— IFEP-RSA and IFDP-RSA (for IFES)
— IFEP-OU and IFDP-OU (for IFES-EPOC)

The choice among DL, EC, and IF primitives affects security to the extent that the difficulties of the underlying problems differ (see D.4). From a practical perspective, the choice of primitive within a family does not significantly affect security against general attacks, because solving the underlying problem is the best currently known general method for attacking the schemes provided that recommended additional methods are employed (although as discussed in D.4.3, the underlying problem for OU is different than for RSA). The choice of -DH vs. -DHC affects security with respect to key validation (see D.5.1.6). The security analysis may vary among the choices; also, certain specific attacks may be relevant for some choices but not for others.

Replace D.5.3.2 with the following text (new title):

D.5.3.2 Additional methods

D.5.3.2.1 Message-encoding methods for encryption

The recommended encoding methods for encryption are EME1 (for IFES) and EME2 and EME3 (for IFES-EPOC). DL/ECIES does not use an encoding method.

An encoding method for IFES should have the following properties, stated informally:

— Representatives of different messages should be unrelated.
— The encoding method, through incorporation of randomness or otherwise, should ensure that representatives of the same message produced at different times are unrelated, and that it is difficult to determine (without the private key), given a ciphertext and a plaintext, whether the ciphertext is an encryption of the plaintext.
— Message representatives should have some verifiable structure, so that it is difficult to produce a ciphertext that decrypts to a valid message representative.
— The encoding method should have minimal mathematical structure that could interact with the encryption primitive (e.g., the encoding method should not be multiplicative, because IFEP-RSA is).

EME1 is considered to have these properties for IFES (one motivation for using EME1 as opposed to other encoding methods is a result by Bleichenbacher [B23]). (See Note 1 for recent results on the security of EME1.)

An encoding method for IFES-EPOC (which produces a randomizer as well as a message representative) should have the same properties except for the third ("verifiable structure"), which is provided directly by the encryption and decryption primitives in IFES-EPOC. It should also ensure that the randomizer is unpredictable.

The three encoding methods also have the additional property of securely associating optional encoding parameters with a message, where the encoding parameters are not encrypted but are protected from modification; see D.5.3.3.

The security of the three encoding methods depends on the underlying hash function (see D.5.2.3.3) and on the length and randomness of the seed (Note 2). The security of EME3 also depends on the method for combining the seed with the message (Note 3).

NOTES

1—*EME1 security:* Bellare and Rogaway's original paper on Optimal Asymmetric Encryption Padding (OAEP) [B18] introduced and gave a security analysis for an encryption scheme based on a predecessor to the EME1 encoding method and an arbitrary trapdoor one-way function *f*. The security analysis in the paper is easily seen to address so-called "indifferent" or "lunch-time" chosen-ciphertext attacks in the random oracle model. However, their paper did not contain a rigorous security analysis for the stronger class of *adaptive* chosen-ciphertext attacks. Two recent results have filled this gap. First, Shoup [Sho01a] has given evidence that a generic security reduction *cannot* be obtained in this sense for an arbitrary *f* (although it is possible to obtain one if the encoding method is modified slightly). Second, Fujisaki et al. [FOP01] have given a security analysis for adaptive chosen ciphertext attacks when *f* is the RSA function. This analysis applies to the EME1 encoding method as well.

An attacker cannot gain any information from an implementation of IFES decryption by observing whether a ciphertext results in an error message, because of the "plaintext awareness" of OAEP. (Informally, the attacker needs to "know" the plaintext in order to construct a valid ciphertext, so the attacker will already know whether a ciphertext will result in an error message.) However, it is important that the implementation output the same error message at the same time regardless of the cause of error. In particular, an implementation should not reveal whether the error is due to the message representative being too long (that is, longer than $\lfloor l/8 \rfloor$ octets) in the EME1 decoding operation. Otherwise an opponent will be able to determine whether the most significant octet of an RSA decryption is nonzero, which may lead to an attack (see Manger [Man01]).

2—*EME2 security:* A security analysis for the IFES-EPOC encryption scheme based on the EME2 encoding method against adaptive chosen-ciphertext attacks has been presented in the random oracle model under the *p*-subgroup assumption. This assumption is, stated roughly, that it is difficult to distinguish $\exp(v, r_0) \bmod n$ and $u \times \exp(v, r_1) \bmod n$, where r_0 and r_1 are uniformly and independently selected, and *n*, *u*, and *v* are as in 8.1.3.3.

The security analysis for IFES-EPOC combines two other analyses: the semantic security of the IFEP-OU primitive (under the *p*-subgroup assumption) given in [OU98] and the security of the EME2 encoding method against adaptive chosen-ciphertext attacks (in the random oracle model) given in [FO99a].

Let (u/n) be the Jacobi symbol of *u* over *n*. If $(u/n) = -1$ and $(v/n) = 1$, the *p*-subgroup assumption is not true. To avoid such a vulnerability, it is recommended that $(u/n) = (v/n)$. This is achieved if the value $v = u^n \bmod n$, i.e., $v_0 = u$ and $v = v_0^n \bmod n$ as noted in 8.1.3.3.

For high assurance of the *p*-subgroup assumption, the length in bits, *rBits*, of the randomized hash value should be $2k+c$ for a positive constant *c* (e.g., $c = 32$) where *k* is the length of the prime factors of the OU modulus *n*.

3—*EME3 security:* A security analysis for the IFES-EPOC encryption scheme based on the EME3 encoding method has been presented in the random oracle model, under the assumption that factoring is difficult for moduli of the form p^2q. For the case that $rBits = \lfloor \log_2 n \rfloor - 1$ where *rBits* is the length in bits of the randomized hash value (this is the maximum length because the randomized hash value must be less than the OU modulus *n*), the security analysis combines two other analyses: the one-wayness of the IFEP-OU primitive (under the factoring assumption) given in [OU98] and the security of the EME3 encoding method against adaptive chosen-ciphertext attacks (in the random oracle model) given in [FO99b].

The case that $rBits = 2k+c$ for a positive constant *c* (e.g., $c = 32$) is analyzed in [NTT01]. In the analysis, a typical case of *v* such as $v = \exp(u, n) \bmod n$ is adopted for simplicity.

4—*Seed length and randomness:* The security of the three encoding methods depends on the unpredictability of the seed.

- For EME1, the seed length equals the length of the hash function output, and the seed value is generated at random. This makes it unlikely both that the seed value can be predicted and that the same seed value will be selected more than once. (If a seed value is selected twice for the same message, for instance, then an opponent will be able to determine that the resulting ciphertexts are for the same message.)
- For EME2, the seed length is recommended to be equal to the hash function output length and the seed value is generated at random for similar reasons.
- For EME3, the seed length is based on the message representative length, which is in turn related to the length of prime factor *p* in the OU private key. For the recommended sizes, the seed length will be at least as long as the hash function output length. The seed value is again generated at random. Random generation of a sufficiently long seed is particularly important in EME3 because the message is encrypted with a symmetric key derived from the seed value.

An adequate security level may also be obtained for these encoding methods if the seed length is shorter (for EME2, where the length is an option), or it is not generated entirely at random, but such choices require further security analysis by the implementer.

5—*Method for combining the seed with the message in EME3:* Except for relatively short messages (such as symmetric keys), the method for combining the seed with the message in EME3 should be a key derivation function combined with a symmetric encryption scheme. A stream cipher based on a key derivation function is not recommended other than for such relatively short messages because the design goal of a key derivation function is to produce correspondingly short keys, not to generate an arbitrarily long keystream; further security analysis is needed if a key derivation is employed as a stream cipher to encrypt messages of arbitrary length. See D.5.3.2.2 and D.5.2.3.5 for security considerations related to key derivation functions and symmetric encryption schemes respectively.

D.5.3.2.2 Key derivation functions

The only recommended key derivation function the encryption schemes in this standard is KDF2, which is an option in EME3 and DL/ECIES.

The security considerations are the same as in the case of key agreement schemes (see D.5.2.2.2). As indicated in D.5.3.2.1, Note 5, a key derivation function is recommended for use as a stream cipher in EME3 only to encrypt relatively short messages. Similar considerations apply to the stream cipher option in DL/ECIES, and furthermore, as explained in D.5.3, Note 1, the messages should be fixed-length for the stream cipher option in non-DHAES mode. For arbitrary length messages, the key derivation function should be used to generate a symmetric key for a symmetric encryption scheme, and the message encrypted with the symmetric encryption scheme.

D.5.3.2.3 Hash functions

The recommended hash functions for the encryption schemes in this standard are SHA-1, RIPEMD-160, SHA-256, SHA-384, and SHA-512, which are options in all three encoding methods for encryption, as well as in KDF2, MGF1, and MAC1, which are used in some of the encryption schemes and encoding methods.

For EME1, collision-resistance is needed if the encoding parameters are a consideration, because they are hashed directly. For EME2 or EME3, this property is not strictly necessary because the message and encoding parameters are hashed together with a random seed, but as noted below, because the hash function instantiates a random oracle, the property is needed for the security analysis.

The properties assumed of a hash function for MAC1 are that it should be difficult to find collisions when the hash function's initialization vector is random and secret, and that it should be difficult to determine the output of the underlying compression function when the initialization vector is random and secret (see Bellare et al. [BCK96]). The first property is weaker than collision resistance, and the second has to do with the "pseudorandomness" of the hash function. The recommended hash functions are considered to have these properties.

A hash function instantiates a random oracle in EME1, EME2, and EME3. As further discussed in D.5.2.2.4 and D.5.2.2.5, no formal properties for hash functions are available that facilitate security analysis outside the random oracle model, but a collision-resistant hash function is a reasonable design approach. For similar reasons, a collision-resistant (and robustly designed) hash function is recommended for constructing a mask generation function.

Hash function properties for use with a key derivation function are discussed in D.5.1.2.2.

D.5.3.2.4 Mask generation functions

The only recommended mask generation function for the encryption schemes in this standard is MGF1, which is an option in EME1. The security considerations are similar to those in the case of signature schemes; see D.5.2.2.5.

D.5.3.2.5 Message authentication codes

The only recommended message authentication code for the encryption schemes in this standard is MAC1, which is an option in DL/ECIES.

A message authentication code for an encryption scheme in this standard should produce authentication tags that are computationally infeasible to generate without the MAC key. MAC1 is considered to have this property, provided that the MAC key and tag are sufficiently long (e.g., each 80 bits or longer). Typical attacks for forging MAC1 authentication tags involve on the order of 2^l operations assuming an l-bit MAC key and an l-bit tag. See Bellare et al. [BCK96] for further discussion.

The security of MAC1 depends on the underlying hash function (see D.5.3.2.3).

D.5.3.2.6 Symmetric encryption schemes

The recommended symmetric encryption scheme for the encryption schemes in this standard are Triple-DES-CBC-IV0 and AES-CBC-IV0, which are options in DL/ECIES and EME3.

The primary attribute of a symmetric encryption scheme for DL/ECIES and EME3 is confidentiality with a one-time key: It should be difficult, given the encrypted message and some information about the message but not the symmetric encryption key, to determine other information about the message. Integrity protection is not strictly necessary because it is otherwise provided by DL/ECIES and EME3. In addition, provided the encryption key in these DL/ECIES and EME3 is a one-time key as recommended, the same message will be encrypted differently at different times, so no randomization is required in the symmetric encryption scheme.

A sufficiently large key size, say, 80 bits or more, is recommended to ensure that brute-force key search is difficult. Triple-DES-CBC-IV0, with an effective key size of at least 112 bits, and AES-CBC-IV0, with at least 128 bits, are considered to provide the necessary properties for DL/ECIES and EME3. (Both schemes are deterministic.) (As also noted in D.5.2.2.6, these are clearly not the only symmetric encryption schemes that may provide adequate security in this context.)

The recommended schemes both allow larger key sizes, which should be considered in applications with higher security requirements (in the case of AES-CBC-IV0, the higher key size involves increased computational effort as well, so there is a tradeoff, but for relatively small messages, this is not likely to be an issue).

Replace D.5.3.3 with the following text (new title):

D.5.3.3 Encoding and key derivation parameters

The purpose of encoding parameters in an encryption scheme is to associate control information with a message. The parameters should have an unambiguous interpretation. Their content depends on the implementation, and they may be omitted (i.e., the parameters may be an empty string). The parameters are not encrypted by the encryption scheme, but they are securely associated with the ciphertext and protected from modification. Whether they have been modified or not is verified during the decryption process. For information on the encoding parameters, see Johnson and Matyas [B86].

As indicated in 11.3.1, any parameters to the symmetric encryption scheme underlying DL/ECIES, such as a variable initialization vector, should be included in the encoding parameters to ensure their integrity, if they are not included in the encrypted message *C* itself.

In the non-DHAES mode of DL/ECIES, consideration should be given to the possibility that the encoding of the input to the MAC, i.e., $C \| P_2$, is ambiguous and may correspond to some other pair $C'' \| P_2'$ where $C \| P_2 = C'' \| P_2'$ (which could make the encryption scheme malleable in a practical sense). There are several straightforward ways to address this concern. For instance, the encoding parameters P_2 might be established in an authentic manner independent of the transmission of *C*. One might also specify that the encoding parameters have a fixed length (perhaps empty), or end with a fixed-length representation of the length of the encoding parameters, or alternatively provide a prefix-free encoding of the message *M* (see D.5.2.2.2, Note 4 for related discussion in another context). In the DHAES mode of DL/ECIES, a fixed-length representation of the encoding parameters is appended, thereby addressing the concern.

Similar to their purpose for key agreement schemes (see D.5.1.4), the key derivation parameters for DL/ECIES influence the selection of the shared secret key. If the sender's key pair is used for additional encryption operations with the same recipient, the key derivation parameters P_1 should vary among the operations so that the key *K* varies, as observed in the note to 11.3.2. The parameters may be the empty string if the sender's key pair is used for only one encryption operation.

As noted in D.5.3, it may be beneficial to include a representation of the sender's public key *v* in the input to the key derivation function, e.g., as part of the key derivation parameters.

For EME1, EME2, and EME3 and for DL/ECIES, the length of the encoding parameters can vary from one encryption operation to another. For DL/ECIES, the length of the key derivation parameters can also vary.

D.5.3.5 Validation of domain parameters and keys

Insert the following paragraphs at the end of the clause:

For the DL/EC encryption schemes, there are additional risks similar to those for the DL/EC key agreement schemes (see D.5.1.6), because the encryption schemes are essentially applications of the key agreement schemes. The additional risks include

— Reduction in size of the key space for derived keys

— Compromise of information about a private key that is combined with the invalid public key in a secret value derivation primitive

The first risk applies to both the sender and the recipient (although the opponent's ability to exploit a reduced key space may vary depending on which party is attacked). The second risk applies primarily to the recipient's private key, because it is typically involved in many encryption operations, but it would also apply to the sender's private key if the sender's private key is involved in more than one encryption operation as suggested in the notes to 11.3.2.

Both risks can be addressed by a number of means, as outlined further in D.5.1.6.

Annex E

(informative)

Formats

E.1 Overview

Change the bulleted list after the second paragraph of the subclause as follows:

— ANSI X9.42 [~~ANS98b~~B8] for DL key agreement
— ANSI X9.63 [~~ANS98f~~B12] for EC key agreement and EC encryption for key transport
— ANSI X9.57 [~~ANS97c~~B6] for DL DSA signatures
— ANSI X9.62 [~~ANS98c~~B11] for EC DSA signatures
— ANSI X9.31 [~~ANS98a~~B7] for IF signatures
— ANSI X9.44 [~~ANS98c~~B9] for IF encryption for key transport

Insert the following paragraph before the last paragraph of the subclause:

Additional examples of ASN.1 syntax can be found in documents such as PKCS #1 [PKCS1v2_1] and SEC-1 [SEC-1]. Alternatives to ASN.1 syntax are also available. In the digital signature specification for the eXtensible Markup Language (XML) (see Eastlake et al. [ERS02]), public keys and digital signatures are represented as ASCII text strings, whereas in Transport Layer Security (TLS) [DA99], various cryptographic values are represented in octet strings with a syntax that is similar to data structures in the C programming language. A recent alternative proposal for representing elliptic curve keys and domain parameters can be found in Schroeppel and Eastlake [SE01].

E.2 Representing basic data types as octet strings

E.2.3 Elliptic curve points (EC2OSP and OS2ECP)

Replace the first sentence with the following text:

Elliptic curve points should be converted to/from octet strings using one of the pairs of primitives defined in 5.5.6.2 or 5.5.6.3. The *x*-coordinate-only representation in 5.5.6.3 should be employed only if the recipient does not need to resolve the ambiguity in the *y*-coordinate, or can do so by other means.

NOTE—In general, the elliptic curve signature schemes in this standard depend on the specific *y*-coordinate, and the elliptic curve key agreement and encryption schemes do not.

Delete the remainder of the subclause, including E.2.3.1 and E.2.3.2.

Replace E.2.4 with the following text (new title):

E.2.4 Polynomials over $GF(p)$, $p \geq 2$ (PN2OSP and OS2PNP)

Polynomials over $GF(p)$, p prime (e.g., field polynomials, when represented as domain parameters) should be converted to/from octet string using primitives PN2OSP and OS2PNP, as defined below.

If $p = 2$, the coefficients of a polynomial $f(t)$ over $GF(2)$ are elements of $GF(2)$ and are, therefore, represented as bits: the element zero of $GF(2)$ is represented by the bit 0, and the element 1 of $GF(2)$ is represented by the bit 1 (see Section 5.5.4). If $p \geq 3$, the coefficients are represented as integers modulo p. Let e be the degree of $f(t)$ and

$$f(t) = a_e t^e + a_{e-1} t^{e-1} + \ldots + a_1 t + a_0$$

where $a_e = 1$. If $p = 2$, to represent $f(t)$ as an octet string, the bits representing its coefficients should be concatenated into a single bit string: $a = a_e \| a_{e\text{-}1} \| \ldots \| a_1 \| a_0$. The bit string a should then be converted into an octet string using BS2OSP (see 5.5.2). If $p \geq 3$, the integer

$$a = a_{t-1} p^{t-1} + \ldots + a_2 p^2 + a_1 p + a_0$$

should first be computed and then converted into an octet string using I2OSP (see 5.5.3).

The primitive that converts polynomials over $GF(p)$ to octet strings is called Polynomial to Octet String Conversion Primitive or PN2OSP. It takes a polynomial $f(t)$ and a prime p as input and outputs an octet string.

The primitive that converts octet strings to polynomials over $GF(p)$ is called Octet String to Polynomial Conversion Primitive or OS2PNP. It takes the octet string and the prime p as inputs and outputs the corresponding polynomial over $GF(p)$. Let l be the length of the input octet string.

If $p = 2$, OS2PNP should use OS2BSP (see 5.5.2) with the octet string and the length $8l$ as inputs. It should output "error" if OS2BSP outputs "error." The output of OS2BSP should be parsed into $8l$ coefficients, 1 bit each. Note that at most seven leftmost coefficients may be zero. The leftmost zero coefficients should be discarded.

If $p \geq 3$, OS2PNP should use OS2IP (see 5.5.3) to obtain the integer a defined above. The resulting polynomial can be obtained from this integer by successively dividing by p and keeping the remainder as in 5.3.3.

NOTE—The representation defined here is intended for arbitrary polynomials. More compact representations are possible for sparse polynomials; see Schroeppel and Eastlake [SE01] for an example.

E.3 Representing outputs of schemes as octet strings

Replace E.3.1 with the following text (new title):

E.3.1 Output data format for DL/EC signature schemes

For DL/ECSSA and DL/ECSSR, the output of the signature generation function (see 10.2.2 and 10.4.2) is a pair of integers (c, d). Let r denote the order of the generator (g or G) in the DL or EC domain parameters, and let $l = \lceil \log_{256} r \rceil$ (i.e., l is the length of r in octets). The output (c, d) may be formatted as an octet string as follows: Convert the integers c and d to octet strings C and D, respectively, of length l octets each, using the primitive I2OSP, and output the concatenation $C \| D$ as the signature. To parse the signature, split the octet string into two components C and D, of length l octets each, and convert them to integers c and d, respectively, using OS2IP. Note that it is essential that both C and D be of length l octets, even if it means that they have leading zero octets.

For DL/ECSSR-PV, the output of the signature generation function (see 10.5.2) is a pair (C, d), where C is an octet string and d is an integer. The output (C, d) may be formatted as an octet string as follows: Convert the integer d to an octet string D of length l octets (where l is as defined above) using the primitive I2OSP, and output the concatenation $C \| D$. To parse the signature, let C be all but the rightmost l octets of the signature and let D be the rightmost l octets, and convert D to an integer d using OS2IP.

NOTE—The output of DL/ECSSA may also be formatted according to the following method, described in more detail in ANSI X9.57-1997 [B6] and ANSI X9.62-1998 [B11]: Combine c and d into an ASN.1 structure (ISO/IEC 8824-1:1998 [B72]), and encode the structure using some encoding rules, such as BER or DER (ISO/IEC 8825-1:1998 [B76]).

Replace E.3.2 with the following text (new title):

E.3.2 Output data format for IF signature schemes

For IFSSA and IFSSR, the output of the signature generation function (see 10.3.2 and 10.6.2) is an integer s. Let $k = \lceil \log_{256} n \rceil$ denote the length in octets of the modulus n in the IF public key. The output s may be formatted as an octet string by simply converting it to an octet string of length k octets using the primitive I2OSP.

Replace E.3.3 with the following text (new title):

E.3.3 Output data format for IF encryption schemes

For IFES, the output of the encryption function (see 11.2.2) is an integer g. Let $k = \lceil \log_{256} n \rceil$ denote the length in octets of the modulus n in the IF public key. The output g may be formatted as an octet string by simply converting it to an octet string of length k octets using the primitive I2OSP.

For IFES-EPOC, the output of the encryption function (see 11.4.2) is a pair (g, C) where g is an integer and C is an optional octet string. The output (g, C) may be formatted as an octet string as follows: Convert the integer g to an octet string G of length k octets (where k is as defined above) using the primitive I2OSP, and output the concatenation $G \| C$ as the ciphertext (if G is absent, then the ciphertext is simply C). To parse the ciphertext, let C be all but the leftmost k octets of the ciphertext and let G be the leftmost k octets, and convert G to an integer g using OS2IP.

Insert the following subclause after E.3.3:

E.3.4 Output data format for DL/EC encryption scheme

For DL/ECIES, the output of the encryption function (see 11.3.2) is a triple (*V, C, T*) where *V* is an octet string, *C* is a bit string, and *T* is a bit string. If the lengths of the bit strings *C* and *T* are both divisible by 8, then the output (*V, C, T*) may be formatted as an octet string as follows: Convert the bit strings *C* and *T* to octet strings *CO* and *TO*, respectively, using B2OSP, and output $V \| CO \| TO$ as the ciphertext. In the DL case, the length of the octet string *V* will be $\lceil \log_{256} q \rceil$ where *q* is the field size; in the EC case, the length will be either $1 + \lceil \log_{256} q \rceil$ or $1 + 2 \lceil \log_{256} q \rceil$ depending on whether the compressed or uncompressed/hybrid format is selected.

To parse the ciphertext, let *V* be the leftmost $\lceil \log_{256} q \rceil$ octets in the DL case and the leftmost $1 + \lceil \log_{256} q \rceil$ or $1 + 2 \lceil \log_{256} q \rceil$ octets in the EC case (depending on the format, which can be determined by examining the first octet). Let *TO* be the rightmost *tBits*/8 octets (where *tBits* is the length of the authentication tag, which is a parameter to the encoding method), and let *CO* be the remaining middle octets. Let *l* be the length of *CO* in octets. Convert *CO* and *TO* to bit strings *C* of length 8*l* bits and *T* of length *tBits*, respectively, using OS2BSP.

NOTE—In ANSI X9.63 [B12], the ciphertext is formatted as a bit string $V \| C \| T$. The format here produces the equivalent octet string when the lengths of the bit strings *C* and *T* are both divisible by eight. In the general case, an octet string may be obtained by converting the bit string $V \| C \| T$ to an octet string of comparable length. However, to parse such a ciphertext, the recipient will need to know how much padding was added in the conversion to the octet string (or, alternatively, the bit length of the encrypted message *C*), so that the end of the tag *T* can be identified.

Insert the following annex after Annex E:

Annex F

(informative)

Information about patents

The following URL, http://standards.ieee.org/db/patents/index.html, provides a listing of standards-related patents.

Annex G

(informative)

Bibliography

EDITOR'S NOTE
For ease of editing, citations to new references in this draft are given in author-year style (e.g., [ABR98]) or by document name (e.g., [ANSI-X9.71]) rather than the IEEE style (e.g., [B8]). The citations should be converted to IEEE style before publication. Citations to some of these references in the body of the document should be updated as well (e.g., FIPS PUB 140-1 [B54] becomes FIPS PUB 140-2 [B54]; Solo et al. [B142] becomes Housley et al. [B142]) when this amendment is merged into IEEE Std 1363-2000.

Delete [B1].

Replace [B8] and [B9] with the following text:

[B8] ANSI X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[B9] ANSI X9.44-2003 (draft), Public Key Cryptography for the Financial Services Industry: Key Establishment Schemes Using Integer Factorization Cryptography, working draft.

Replace [B12] and [B13] with the following text:

[B12] ANSI X9.63-2002, Public Key Cryptography for the Financial Services Industry: Key Agreement and Transport Using Elliptic Curve Cryptography.

[B13] ANSI X9.80-2001, Public Key Cryptography for the Financial Services Industry: Prime Number Generation, Primality Testing, and Primality Certificates.

Replace [B54] to [B57] with the following text:

[B54] FIPS PUB 140-2, Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication 140-2, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, VA, May 25, 2001 (supersedes FIPS PUB 140-1). Available at http://csrc.nist.gov/fips/

[B55] FIPS PUB 180-2, Secure Hash Standard, Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, VA, August 1, 2002 (supersedes FIPS PUB 180-1). Available at http://csrc.nist.gov/fips/

[B56] FIPS PUB 186-2, Digital Signature Standard, Federal Information Processing Standards Publication 186-2, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, VA, January 27, 2000 (supersedes FIPS PUB 186-1). Available at http://csrc.nist.gov/fips/

[B57] Gallant, R., Lambert, R., and Vanstone, S., “Improving the parallelized Pollard lambda search on binary anomalous curves,” *Mathematics of Computation*, vol. 69, pp. 1699–1705, 2000.

Replace [B72] to [B77] with the following text:

[B72] ISO/IEC 8824-1:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. Equivalent to ITU-T Rec. X.680.

[B73] ISO/IEC 8824-2:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Information Object Specification. Equivalent to ITU-T Rec. X.681.

[B74] ISO/IEC 8824-3:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Constraint Specification. Equivalent to ITU-T Rec. X.682.

[B75] ISO/IEC 8824-4:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications. Equivalent to ITU-T Rec. X.683.

[B76] ISO/IEC 8825-1:2002, Information Technology—ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (CER). Equivalent to ITU-T Rec. X.690.

[B77] ISO/IEC 8826-1:2002, Information Technology—ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER). Equivalent to ITU-T Rec. X.691.

Replace [B79] and [B80] with the following text:

[B79] ISO/IEC 9796-3:2000, Information technology—Security techniques—Digital signature schemes giving message recovery—Part 3: Discrete logarithm based mechanisms.

[B80] ISO/IEC 14888-3:1998, Information technology—Security techniques—Digital signatures with appendix—Part 3: Certificate-based mechanisms.

Replace [B82] with the following text:

[B82] ITU-T, Recommendation X.509 (Mar. 2000), Information technology—Open Systems Interconnection—The Directory: Public-key and Attribute Certificate Frameworks, International Telecommunications Union.

Replace [B119] with the following text:

[B119] National Institute of Standards and Technology (NIST), “Recommended Elliptic Curves for Federal Government Use” (draft), 1999. Available at http://csrc.nist.gov/CryptoToolkit/tkdigsigs.html. Published as Appendix 6 of [B56].

Replace [B141] and [B142] with the following text and move them to maintain alphabetical order:

[B141] Myers, M., Adams, C., Solo, D., and Kemp, D., “RFC 2511: Internet X.509 Certificate Request Format,” Internet Activities Board, Mar. 1999. Available at http://www.rfc-editor.org/

[B142] Housley, R., Polk, W., Ford, W., and Solo, D., “RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” Apr. 2002. Available at http://www.rfc-editor.org/

Replace [B144] with the following text:

[B144] Standards for Efficient Cryptography, "SEC2: Recommended Elliptic Curve Domain Parameters," Sept. 2002. Available at http://www.secg.org/secg_docs.htm

Insert the following references in alpha-numeric order and renumber per IEEE style. Update bibliographical citations in text to reflect new numbers:

[ABR98] Abdalla, M., Bellare, M., and Rogaway, P., "DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem," submission to IEEE P1363a, Sept. 1998. Available at http://grouper.ieee.org/groups/1363/. Revised version via URL of authors.

[AF99] Adams, C. and Farrell, S., "RFC 2510: Internet X.509 Public Key Infrastructure Certificate Management Protocols," Mar. 1999. Available at http://www.rfc-editor.org/

[AM95] Adleman, L. M. and McCurley, K. S., "Open problems in number theoretic complexity, II," in L. M. Adleman and M-D. Huang, eds., *Algorithmic Number Theory, ANTS-I, Lecture Notes in Computer Science.* New York: Springer, 1994, pp. 291–322.

[ANSI-X9.52] ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation.

[ANSI-X9.71] ANSI X9.71-2000, Keyed Hash Message Authentication Code (MAC).

[ABMSV03] Antipa, A., Brown, D., Menezes, A., Struik, R., and Vanstone, S., "Validation of elliptic Curve public keys," in Y. G. Desmedt, ed., *Public Key Cryptography, PKC 2003, Lecture Notes in Computer Science.* New York: Springer, 2003, pp. 211–223. Presentation available at http://grouper.ieee.org/groups/1363/

[AHK99] Aoki, K., Hoshino, F., and Kobayashi, T., "OEF using a successive extension," presented at the rump session of CRYPTO '99.

[BP98] Bailey, D. V. and Paar, C., "Optimal extension fields for fast arithmetic in public-key algorithms," in H. Krawcyzk, ed., *Advances in Cryptology, CRYPTO '98, Lecture Notes in Computer Science.* New York: Springer, 1998, pp. 472–485.

[BP00] Bailey, D. V., and Paar, C., "Efficient arithmetic in finite field extensions with application in elliptic curve cryptography," *Journal of Cryptology,* vol. 14, pp. 153–176, 2001.

[BKL+02] Barreto, P. S. L. M., Kim, H. Y., Lynn, B., and Scott, M., "Efficient algorithms for pairing-based cryptosystems," in M. Yung, Ed., *Advances in Cryptology, CRYPTO 2002, Lecture Notes in Computer Science.* New York: Springer, 2002, pp. 354–368. Revised version available at http://eprint.iacr.org/2002/088

[BCK96] Bellare, M., Canetti, R., and Krawczyk, H., "Keyed hash functions and message authentication," in N. Koblitz, ed., *Advances in Cryptology, CRYPTO '96, Lecture Notes in Computer Science*. New York: Springer, 1996, pp. 1–15. Also available at http://www.research.ibm.com/security/keyed-md5.html

[BR93] Bellare, M. and Rogaway, P., "Random oracles are practical: A paradigm for designing efficient protocols," *Proceedings of the First Annual Conference on Computer and Communications Security,* pp. 62–73, 1993. Revised version available via URL of either author.

[Ber01] Bernstein, D. J., "Circuits for Integer Factorization: A Proposal," manuscript, Nov. 9, 2001. Available at http://cr.yp.to/papers.html

[BMM00] Biehl, I., Meyer, B., and Müller, V., “Differential fault attacks on elliptic curve cryptosystems,” in M. Bellare, Ed., *Advances in Cryptology, CRYPTO 2000, Lecture Notes in Compute Science.* New York: Springer, 2000, pp. 131–146.

[Ble01] Bleichenbacher, D., “On the Generation of DSA One-Time Keys,” manuscript.

[VGT88a] Boneh, D., Durfee, G., and Howgrave-Graham, N., “Factoring N = prq for large r,” in M. J. Wiener, Ed., *Advances in Cryptology, CRYPTO '99, Lecture Notes in Computer Science.* New York: Springer, 1999, pp. 326–337.

[BD86] Brickell, E. F., and DeLaurentis, J. M., “An attack on a signature scheme proposed by Okamoto and Shiraishi,” in H. C. Williams, Ed., *Advances in Cryptology, CRYPTO '85, Lecture Notes in Computer Science.* New York: Springer, 1996, pp. 28–32.

[BO91] Brickell, E., and Odlyzko, A,. “Cryptanalysis: A survey of recent results,” in G. J. Simmons, Ed., *Contemporary Cryptology.* New York: IEEE Press, 1991, pp. 501–540.

[BJ00] Brown, D. R. L., and Johnson, D. B., “Formal security proofs for a signature scheme with partial message recovery,” in D. Naccache, Ed., *Cryptographers' Track RSA Conference 2001, Lecture Notes in Computer Science*. New York: Springer, 2001, pp. 126–142.

[CGL98] Canetti, R., Goldreich, O., and Halevi, S., “The random oracle methodology, revisited (preliminary version),” *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 209–218, 1998.

[CHJ99] Coppersmith, D., Halevi, S., and Jutla, C., “ISO 9796-1 and the new forgery strategy,” presented at the rump session of CRYPTO ‘99.

[Cor00] Coron, J-S., “On the exact security of full domain hash,” in M. Bellare, Ed., *Advances in Cryptology, CRYPTO 2000, Lecture Notes in Computer Science*. New York: Springer, 2000, pp. 229–235.

[Cor02] Coron, J-S., “Optimal security proofs for PSS and other signature schemes,” in L. Knudsen, Ed., *Advances in Cryptology, EUROCRYPT 2002, Lecture Notes in Computer Science*. New York: Springer, 2002, pp. 272–287.

[CNS99] Coron, J-S., Naccache, D., and Stern, J. P., “On the security of RSA padding,” in M. J. Wiener, Ed., *Advances in Cryptology, CRYPTO '99, Lecture Notes in Computer Science*. New York: Springer, 1999, pp. 1–18.

[DA99] Dierks, T., and Allen, C., “RFC 2246: The TLS Protocol Version 1.0,” Internet Activities Board, Jan. 1999. Available at http://www.rfc-editor.org/.

[ERS02] Eastlake, D., Reagle, J., and Solo, D., “RFC 3275: XML-Signature Syntax and Processing,” Internet Activities Board, Mar. 2002. Available at http://www.rfc-editor.org/

[ENS01] El Mahassni, E., Nguyen, P. Q., and Shparlinski, I. E., “The insecurity of Nyberg-Rueppel and other DSA-like signature schemes with partially known nonces,” in J. H. Silverman, Ed., *Proceedings of Cryptography and Lattices Conference, CaLC 2001, Lecture Notes in Computer Science.* New York: Springer, 2000, pp. 97–109. Also available at http://www.comp.mq.edu.au/~igor/Publ.html

[FIPS-46-3] FIPS PUB 46-3, Data Encryption Standard (DES), Federal Information Processing Standards Publication 46-3, U.S. Department of Commerce, National Institute of Standards and Technology (NIST), National Technical Information Service, Springfield, Virginia, Oct. 25, 1999 (supersedes FIPS PUB 46-2). Available at http://csrc.nist.gov/fips/

[FIPS-81] FIPS PUB 81, DES Modes of Operation, Federal Information Processing Standards Publication 81, U.S. Department of Commerce, National Institute of Standards and Technology (NIST), National Technical Information Service, Springfield, Virginia, Dec. 2, 1980. Available at http://csrc.nist.gov/fips/

[FIPS-197] FIPS PUB 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, U.S. Department of Commerce/National Institute of Standards and Technology, Nov. 26, 2001. Available at http://csrc.nist.gov/fips/

[FIPS-198] FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198, U.S. Department of Commerce, National Institute of Standards and Technology, Mar. 6, 2002. Available at http://csrc.nist.gov/fips/

[FO98] Fujisaki, E., and Okamoto, T., "Security of efficient digital signature scheme TSH-ESIGN," manuscript, Nov. 1998. (Also available as Appendix A of Okamoto, T., Fujisaki, E., and Morita, H., "TSH-ESIGN: Efficient Digital Signature Scheme Using Trisection Size Hash," submission to IEEE P1363a, Nov. 1998. Available at http://grouper.ieee.org/groups/1363/)

[FO99a] Fujisaki, E., and Okamoto, T., "How to enhance the security of public-key encryption at minimum cost," in H. Imai and Y. Zheng, Eds., *Public Key Cryptography, PKC '99, Lecture Notes in Computer Science.* New York: Springer, 1999, pp. 53–68.

[FO99b] Fujisaki, E., and Okamoto, T., "Secure integration of asymmetric and symmetric encryption schemes," in M. J. Wiener, Ed., *Advances in Cryptology, CRYPTO '99, Lecture Notes in Computer Science.* New York: Springer, 1999, pp. 537–554.

[FOP01] Fujisaki, E., Okamoto, T., Pointcheval, D., and Stern, J., "RSA-OAEP is secure under the RSA assumption," in J. Kilian, Ed., *Advances in Cryptology, CRYPTO 2001, Lecture Notes in Computer Science*. New York: Springer, 2001, pp. 260–274.

[GHS02] Gaudry, P., Hess, F., and Smart, N., "Constructive and destructive facets of Weil descent on elliptic Curves," *Journal of Cryptology,* vol. 15, pp. 19–46, 2002. Also available at http://ultralix.polytechnique.fr/Labo/Pierrick.Gaudry/papers.html

[GTV88] Girault, M., Toffin, P., and Vallée, B., "Computation of approximate L-th roots modulo n and application to cryptography," in S. Goldwasser, Ed., *Advances in Cryptology, CRYPTO '88, Lecture Notes in Computer Science*. New York: Springer, 1990, pp. 100–117.

[Hou02] Housley, R., "RFC 3370: Cryptographic Message Syntax (CMS) Algorithms." Internet Activities Board, Aug. 2002. Available at http://www.rfc-editor.org/

[HS01] Howgrave-Graham, N. A., and Smart, N., "Lattice attacks on digital signature schemes," *Designs, Codes and Cryptography,* vol. 23, pp. 283–290, Aug. 2001.

[ISO-9796-2] ISO/IEC 9796-2:1997, Information technology-Security techniques-Digital signature schemes giving message recovery—Part 2: Mechanisms using a hash-function.

[ISO-9796-2-revision] ISO/IEC FDIS 9796-2, Information technology—Security techniques—Digital signature scheme giving message recovery—Part 2: Integer factorization based mechanisms, draft, Dec. 16, 2001. (Revision of ISO/IEC 9796-2:1997.)

[ISO-10118-3] ISO/IEC 10118-3:1998, Information technology—Security techniques—Hash-functions—Part 3: Dedicated hash-functions.

[ISO-15946-4] ISO/IEC FCD 15946-4, Information technology—Security techniques—Cryptographic techniques based on elliptic curves—Part 4: Digital signatures with message recovery, draft, May 20, 2002.

[JQ01] Joye, M., and Quisquater, J-J., "On Rabin-type signatures," in B. Honary, Ed., *Cryptography and Coding, Lecture Notes in Computer Science*. New York: Springer, 2001, pp. 99–113.

[Kal02] Kaliski, B., "On hash function firewalls in signature schemes," in B. Preneel, Ed., *Cryptographers' Track RSA Conference 2002, Lecture Notes in Computer Science*. New York: Springer, 2001, pp. 1–16.

[KL99] Kaliski, B. S., Jr., and Liskov, M,. "Efficient finite field basis conversion involving dual bases," in C. K. Koç and C. Paar, Eds., *Cryptographic Hardware and Embedded Systems, CHES '99, Lecture Notes in Computer Science*. New York: Springer, 1999, pp. 135–143.

[KS98] Kaliski, B., and Staddon, J., "RFC 2437: PKCS #1: RSA Cryptography Specifications Version 2.0," Internet Activities Board, Oct. 1998. Available at http://www.rfc-editor.org/.

[KY98] Kaliski, B. S., Jr., and Yin, Y. L., "Storage-efficient finite field basis conversion," in S. Tavares and H. Meijer, Eds., *Selected Areas in Cryptography, SAC '98, Lecture Notes in Computer Science*. New York: Springer, 1999, pp. 81–93.

[Kob98] Koblitz, N. *Algebraic Aspects of Cryptography.* New York: Springer, 1998.

[KBC97] Krawczyk, H., Bellare, M., and Canetti, R., "RFC 2104: HMAC: Keyed-Hashing for Message Authentication," Internet Activities Board, Feb. 1997. Available at http://www.rfc-editor.org/

[LS03] Leadbitter, P. J., and Smart, N. "Cryptanalysis of MQV with partially known nonces," 6th Information Security Conference, ISC 2003, Springer, to appear. Earlier version available from http://eprint.iacr.org/2002/145/

[Len97] Lenstra, A. K. "Using Cyclotomic Polynomials to Construct Efficient Discrete Logarithm Cryptosystems over Finite Fields," V. Varadharajan, J. Pieprzyk, and Y. Mu, Eds., *Information Security and Privacy, ACISP '97, Lecture Notes in Computer Science.* New York: Springer, 1997, pp. 127–138.

[LST+02] Lenstra, A. K., Shamir, A., Tomlinson, J., and Tromer, E., "Analysis of Bernstein's factorization circuit," in Y. Zheng, Ed., *Advances in Cryptology, ASIACRYPT 2002, Lecture Notes in Computer Science.* New York: Springer, 2002, pp. 1–26. Also available via http://www.wisdom.weizmann.ac.il/~tromer/

[LN94] Lidl, R., and Niederreiter, H., *Introduction to Finite Fields and their Applications*. Cambridge, U.K.: Cambridge University Press, 1994.

[Man01] Manger, J., "A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0," in J. Kilian, Ed., *Advances in Cryptology, CRYPTO 2001, Lecture Notes in Computer Science.* New York: Springer, 2001, pp. 230–238.

[Mih97] Mihailescu, P., "Optimal Galois bases which are not normal," presented at Fast Software Encryption rump session, FSE '97, Haifa, Israel, Jan. 20–22, 1997.

[MNP+98] M'Raïhi, D., Naccache, D., Pointcheval, D., and Vaudenay, S., "Computational alternatives to random number generators," in S. Tavares and H. Meijer, Eds., *Selected Areas in Cryptography, SAC '98, Lecture Notes in Computer Science.* New York: Springer, 1999, pp. 72–80.

[MLSW00] Myers, M., Liu, X., Schaad, J., and Weinstein, J., "IETF RFC 2797: Certificate Management Messages over CMS," Apr. 2000. Available at http://www.rfc-editor.org/

[NMV+95] Naccache, D., M'Raïhi, D., Vaudenay, S., and Raphaeli, D., "Can D.S.A. be improved? Complexity trade-offs with the digital signature standard," in A. De Santis, Ed., *Advances in Cryptology, EUROCRYPT '94, Lecture Notes in Computer Science.* New York: Springer, 1995, pp. 77–85.

[NIST03a] National Institute of Standards and Technology, NIST Special Publication 800-56: Recommendation on Key Establishment Schemes. Draft 2.0, Jan. 2003. Available at http://csrc.nist.gov/

[NIST03b] National Institute of Standards and Technology, NIST Special Publication 800-57: Recommendation for Key Management, Part 1: General Guideline. Draft, Jan. 2003. Available at http://csrc.nist.gov/

[NIST03c] National Institute of Standards and Technology, NIST Special Publication 800-57: Recommendation for Key Management, Part 2: Best Practices for Key Management Organization. Draft, Jan. 2003. Available at http://csrc.nist.gov/

[Ngu01] Nguyen, P. Q., "The dark side of the hidden number problem: Lattice attacks on DSA," in K. Lam, I. Shparlinski,, H. Wang, and C. Xing, Eds., *Cryptography and Computational Number Theory, CCNT '99, Progress in Computer Science and Applied Logic*. Berlin: Birkhäuser, 2001, pp. 321–330.

[NS02] Nguyen, P. Q., and Shparlinski, I. E., "The insecurity of the digital signature algorithm with partially known nonces," *Journal of Cryptology*, vol. 15, pp. 151–176, 2002. Available at http://www.comp.mq.edu.au/~igor/Publ.html

[NS03] Nguyen, P. Q., and Shparlinski, I. E., "The insecurity of the elliptic curve digital signature algorithm with partially known nonces," *Designs, Codes and Cryptography*, vol. 30, pp. 201–217, 2003. Available at http://www.comp.mq.edu.au/~igor/Publ.html

[NTT01] NTT, Self-Evaluation of EPOC-2: (Revised) Submission to NESSIE. 2001. Available at http://info.isl.ntt.co.jp/epoc/

[Oka90] Okamoto, T. "A fast signature scheme based on congruential polynomial operations," *IEEE Transactions on Information Theory*, vol. 36, pp. 47–53, 1990.

[OU98] Okamoto, T., and Uchiyama, S., "A new public-key cryptosystem as secure as factoring," in K. Nyberg, Ed., *Advances in Cryptology, EUROCRYPT '98, Lecture Notes in Computer Science.* New York: Springer, 1998, pp. 308–318.

[PKCS1v2_1] Public-key Cryptography Standards (PKCS), PKCS #1 v2.1: RSA Cryptography Standard, 2002. Available at http://www.rsasecurity.com/rsalabs/pkcs/

[Per97] Peralta, R., "Bleichenbacher's improvement for factoring numbers of the form N = P2Q," unpublished communication, 1997.

[PO96] Peralta, R., and Okamoto, E., "Faster factoring of integers of a special form," *IEICE Transactions on Fundamentals*, vol. E79-A, pp. 489–493, 1996.

[PV00] Pintsov, L. A., and Vanstone, S. A., "Postal revenue collection in the digital age," in Y. Frankel, Ed., *Financial Cryptography, FC '00, Lecture Notes in Computer Science.* New York: Springer, 2000, pp. 105–120.

[Pol97] Pollard, J. L., Manuscript, 1997.

[PKCS-5-v1_5] Public-Key Cryptography Standards (PKCS), PKCS #5 v1.5: Password-Based Encryption Standard, Nov. 1, 1993. Available at http://www.rsasecurity.com/rsalabs/pkcs/

[Res99] Rescorla, E., “RFC 2631: Diffie-Hellman Key Agreement Method,” Internet Activities Board, June 1999. Available at http://www.rfc-editor.org/

[Rie94] Riesel, H.. *Prime Numbers and Computer Methods for Factorization*, 2d ed. Berlin: Birkhäuser, 1994.

[SWD96] Schirokauer, O., Weber, D., and Denny, T., “Discrete logarithms: The effectiveness of the index calculus method,” in H. Cohen, Ed., *Algorithmic Number Theory, ANTS-II, Lecture Notes in Computer Science.* New York: Springer, 1996, pp. 337–361.

[SE02] Schroeppel, R. C., and Eastlake, D., 3rd., “ECC KEYs in the DNS,” Internet-Draft http://www.ietf.org/internet-drafts/draft-ietf-dnsext-ecc-key-02.txt, May 2002 (work in progress, expires Nov. 2002).

[SEC-1] Standards for Efficient Cryptography, SEC1: Elliptic Curve Cryptography, v1.0, Sept. 20, 2000. Available at http://www.secg.org/

[ST03] Shamir, A., and Tromer, E., “Factoring large numbers with the TWIRL device,” in D. Boneh, Ed., *Advances in Cryptology, CRYPTO 2003, Lecture Notes in Computer Science.* New York: Springer, 2003, pp. 1–26. Available via http://www.wisdom.weizmann.ac.il/~tromer/

[Shi01] Shipsey, R., ESIGN, NESSIE public report NES\DOC\RHU\WP3\005\b, Jan. 31, 2001. Available at https://www.cosic.esat.kuleuven.ac.be/nessie/reports/rhuwp3-005b.pdf

[Sho01a] Shoup, V., “OAEP reconsidered,” in J. Kilian, Ed., *Advances in Cryptology, CRYPTO 2001, Lecture Notes in Computer Science*. New York: Springer, 2001, pp. 239–259.

[Sho01b] Shoup, V., “A Proposal for an ISO Standard for Public Key Encryption (version 2.1),” Dec. 20, 2001. Available at http://shoup.net/

[Sim94] Simmons, G. J., “Subliminal communications is easy using the DSA,” in T. Helleseth, Ed., *Advances in Cryptology, EUROCRYPT ‘93, Lecture Notes in Computer Science*. New York: Springer, 1994, pp. 218–232.

[VGT88a] Vallée, B., Girault, M., and Toffin, P., “How to break Okamoto’s cryptosystem by reducing lattice bases,” in C. G. Günther, Ed., *Advances in Cryptology, EUROCRYPT '88, Lecture Notes in Computer Science.* New York: Springer, 1988, pp. 281–291.

[VGT88b] Vallée, B., Girault, M., and Toffin. P., “How to guess L-th roots modulo n by reducing lattice bases,” in T. Mora, Ed., *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC-6 and ISSAC ‘88, Lecture Notes in Computer Science*. New York: Springer, 1988.