IEEE STANDARDS ASSOCIATION

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

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

IEEE Std 1363.3™-2013

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

Sponsor

Microprocessor Standards Committee
of the
IEEE Computer Society

Approved 22 August 2013

IEEE-SA Standards Board

Abstract: Common identity-based public-key cryptographic techniques that use pairings, including mathematical primitives for secret value (key) derivation, public-key encryption, and digital signatures, as well as cryptographic schemes based on those primitives are specified in this standard. Also, related cryptographic parameters, public keys and private keys, are specified. The purpose of this standard is to provide a reference for specifications of a variety of techniques from which applications may select.
Keywords: encryption, identity-based encryption, IEEE 1363.3™, pairing-based cryptography, pairing-based encryption, public-key cryptography

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

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

Important Notices and Disclaimers Concerning IEEE Standards Documents

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

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

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

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

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

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

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

Translations

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

Official statements

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

Comments on standards

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

Comments on standards should be submitted to the following address:

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

Laws and regulations

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

Copyrights

Photocopies

Updating of IEEE Standards documents

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

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

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

Errata

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

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

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

Participants

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

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

Kendall Ananyi
Matt Ball
Xavier Boyen
Mike Brenner
Daniel Brown
Mark Chimley
Andy Dancer
Mike Geipel
David Jablon
Satoru Kanno
Tetsutaro Kobayashi
David Kravitz
Phil MacKenzie
Michael Markowitz
Luther Martin
Marc Provencher
Jim Randall
Roger Schlafly
Mike Scott
Hovav Shacham
Ari Singer
Terence Spies
Yongge Wang
Tom Wu
Go Yamamoto

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

Ed Addario
Mike Brenner
Keith Chow
Andy Dancer
James Davis
Thomas Dineen
Andrew Fieldsend
Randall C. Groves
Werner Hoelzl
Atsushi Ito
Mark Jaeger
Piotr Karocki
Thomas Kurihara
Susan Land
William Lumpkins
Greg Luri
Michael S. Newman
Nick S.A. Nikjoo
Randall Safier
Bartien Sayogo
Gil Shultz
Kapil Sood
Thomas Starai
Rene Struik
Walter Struppler
Joseph Tardo
Srinivasa Vemuru
John Vergis
Karl Weber
William Whyte
Oren Yuen
Janusz Zalewski

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

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

Satish Aggarwal
Masayuki Ariyoshi
Peter Balma
William Bartley
Ted Burse
Clint Chaplin
Wael Diab
Jean-Philippe Faure
Alexander Gelman
Paul Houzé
Jim Hughes
Young Kyun Kim
Joseph L. Koepfinger*
David J. Law
Thomas Lee
Hung Ling
Oleg Logvinov
Ted Olsen
Gary Robinson
Jon Walter Rosdahl
Mike Seavey
Yatin Trivedi
Phil Winston
Yu Yuan

*Member Emeritus

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

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

Don Messina
IEEE Standards Program Manager, Document Development

Joan Woolery
IEEE Client Services Manager, Professional Services

Introduction

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

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

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

Contents

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings

1. Overview

1.1 Scope

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

1.2 Purpose

The proliferation of electronic communication and the Internet brings with it the need for privacy and data protection. Public-key cryptography offers fundamental technology addressing this need. Many alternative public-key techniques have been proposed, each with its own benefits. IEEE Std 1363™-2000[1] and IEEE Std 1363a™-2004 have produced a comprehensive reference defining a range of common public-key techniques covering key agreement, public-key encryption, and digital signatures from several families, namely the discrete logarithm, integer factorization, and elliptic curve families. This document will specify identity-based cryptographic techniques based on pairings. These offer advantages over classic public-key

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

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

1.3 Organization of the document

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

1.3.1 Structure of the main document

The structure of the main document is as follows:

- Clause 1 is a general overview.
- Clause 2 provides references to other standards and publications.
- Clause 3 defines some terms used throughout this standard.
- Clause 4 gives an overview of the types of cryptographic techniques that are defined in this standard.
- Clause 5 defines certain mathematical conventions used in the standard, including notation and representation of mathematical objects. It also defines formats to be used in communicating the mathematical objects as well as primitives for data type conversion.
- Clause 6 defines certain cryptographic hashing primitives that are used to build the complete schemes described in subsequent clauses.
- Clause 7 defines certain pairing-based primitives that are used to build the identity-based schemes that are defined in the subsequent clauses.
- Clause 8 defines four pairing-based, identity-based encryption schemes.
- Clause 9 defines a pairing-based, identity-based signature scheme.
- Clause 10 defines a pairing-based, identity-based signcryption scheme.
- Clause 11 defines two pairing-based, identity-based key agreement schemes.

1.3.2 Structure of the annexes

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

- Annex A (informative) Number-theoretic background
- Annex B (normative) Conformance
- Annex C (informative) Rationale
- Annex D (informative) Security considerations
- Annex E (informative) Formats
- Annex F (informative) Bibliography

2. Normative references

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

IEEE Std 1363™-2000, IEEE Standard Specifications for Public-Key Cryptography.[2, 3]

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

3. Definitions

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

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

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

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

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

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

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

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

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

[2] IEEE publications are available from The Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).
[3] The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.
[4] *IEEE Standards Dictionary: Glossary of Terms and Definitions* is available at http://shop.ieee.org.

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

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

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

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

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

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

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

finite field: A field in which there are only a finite number of quantities. The discrete logarithm (DL) and elliptic curve (EC) schemes are always implemented over finite fields. See Clause 5 for a description of the particular finite fields used in this standard.

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

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

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

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

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

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

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

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

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

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

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

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

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

length: (A) Length of a bit string is the number of bits in the string. **(B)** Length of an octet string is the number of octets in the string. **(C)** Length in bits of a nonnegative integer n is $\lceil \log_2(n+1) \rceil$ (i.e., the number of bits in the integer's binary representation). **(D)** Length in octets of a nonnegative integer n is $\lceil \log_{256}(n+1) \rceil$ (i.e., the number of digits in the integer's representation base 256). For example, the length in bits of the integer 500 is 9, whereas its length in octets is 2.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Types of cryptographic techniques

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

4.1 General model

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

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

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

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

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

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

A general framework of primitives is provided in Clause 5 through Clause 7; specific schemes are defined in Clause 8 through Clause 11. This standard, however, does not define protocols. These are application-specific and hence are outside the scope of the standard. Nevertheless, the techniques defined in this standard are key components for constructing various cryptographic protocols. Also, Annex D discusses security considerations related to how the techniques can be used in protocols to achieve certain security attributes.

4.2 Primitives

The following types of primitives are defined in this standard:

— Cryptographically hashing an octet string to either an integer or a point on an elliptic curve

— Generation of an identity-based private key

— Verification of an identity-based private key

— Encryption of a plaintext message

— Decryption of a ciphertext message

— Generation of an identity-based signature

— Verification of an identity-based signature

— Reencryption of a ciphertext into a different ciphertext

— Derivation of an identity-based public key

— Derivation of a shared secret from public and private keys

— Encapsulation of a random session key

— Decapsulation of an encrypted session key

— Setup of system parameters needed for the operation of an identity-based cryptographic scheme

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

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

The specification of a primitive consists of the following information:

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

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

4.3 Schemes

The following types of schemes are defined in this standard:

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

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

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

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

The specification of a scheme consists of the following information:

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

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

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

4.4 Table summary

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

Table 1—Summary of schemes and their components

Scheme name	Components	Primitives
SK KEM	SK-KEM-S SK-KEM-EX SK-KEM-EN SK-KEM-DE	P-SK-G P-SK-V P-SK-E P-SK-D IHF1 SHF1
BB_1 KEM	BB1-KEM-S BB1-KEM-EX BB1-KEM-EN BB1-KEM-DE	P-BB1-G P-BB1-V P-BB1-E P-BB1-D IHF1 SHF1
BB_1 IBE	BB1-IBE-S BB1-IBE-EX BB1-IBE-EN BB1-IBE-DE	P-BB1-G P-BB1-V P-BB1-E P-BB1-D IHF1 SHF1
BF IBE	BF-IBE-S BF-IBE-EX BF-IBE-EN BF-IBE-DE	P-BF-G P-BF-V P-BF-E P-BF-D IHF1 SHF1 PHF-SS PHF-GFP PHF-GF2
BLMQ signature	BLMQ-SIG-S BLMQ-SIG-EX BLMQ-SIG-SI BLMQ-SIG-VE	P-BLMQ-G P-BLMQ-GV
BLMQ signcryption	BLMQ-SC-S BLMQ-SC-EX BLMQ-SC-SE BLMQ-SC-DV	P-BLMQ-G P-BLMQ-GV
Wang key agreement	WKA-KA-D1 WKA-KA-D2 WKA-KA-V WKA-KA-D3 WKA-KA-G	P-BF-D P-BF-G PHF-SS PHF-GFP PHF-GF2
SCC key agreement	SCC-KA-G	P-BF-G P-BF-V P-SCC-D1

5. Mathematical conventions

This clause describes certain mathematical conventions used in the standard, including notation, terminology, and representation of mathematical objects. It also contains primitives for data type conversion. Note that the internal representation of mathematical objects is left entirely to the implementation and may or may not follow the one described in 5.1.

5.1 Mathematical notation

The following mathematical notation is used throughout the document.

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

1 denotes the integer 1, the bit 1, or the multiplicative identity (the element one) of a finite field. See 5.3 for more on finite fields.

$a \times b$ denotes the product of *a* and *b*, where *a* and *b* are either both integers or both finite field elements. When it does not cause confusion, $\times$ is omitted and the notation *ab* is used. See 5.3 for more on finite fields.

$a \times P$ denotes scalar multiplication of an elliptic curve point *P* by a non-negative integer *a*. When it does not cause confusion, $\times$ is omitted and the notation *aP* is used. See 5.4 for more on elliptic curves.

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

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

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

$LCM(a,b)$ denotes the least common multiple of *a* and *b* for two positive integers *a* and *b* (i.e., the least positive integer that is divisible by both *a* and *b*). See A.1.1 and A.2.2 for an algorithm to compute the LCM.

$GCD(a,b)$ denotes the greatest common divisor of *a* and *b* for two positive integers *a* and *b* (i.e., the largest positive integer that divides both *a* and *b*). See A.1.1 and A.2.2 for an algorithm to compute the GCD.

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

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

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

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

$a \bmod n$ denotes the unique remainder r, $0 \le r < n$, when the integer a is divided by the positive integer n. For example, 23 mod 7 = 2. The operator "mod" has the lowest precedence of all arithmetic operators (e.g., 5 + 8 mod 3 is equal to 13 mod 3, not 5 + 2). See A.1.1 for more details.

$a \equiv b (\bmod n)$ denotes that $n \mid (a - b)$, so that the integers a and b have the same remainder when divided by the positive integer n. It is pronounced "a is congruent to b modulo n." This is equivalent to (a mod n) = (b mod n). See A.1.1 for more details.

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

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

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

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

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

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

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

$E^t / GF(q)$ denotes the order twist of order t of the elliptic curve $E / GF(q)$.

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

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

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

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

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

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

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

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

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

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

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

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

5.2 Bit strings and octet strings

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

NOTE—When a string is represented as a sequence, it may be indexed from right to left or from left to right, starting with any index; this does not change the meaning of the terms above. For example, consider the octet string of 4 octets: 1c 76 3b e4. One can represent it as a string a0 a1 a2 a3 with a0 = 1c, a1 = 76, a2 = 3b, and a3 = e4. In this case, a0 represents the first octet, and a3 represents the last octet. Alternatively, one can represent it as a string a1 a2 a3 a4 with a1 = 1c, a2 = 76, a3 = 3b, and a4 = e4. In this case, a1 represents the first octet, and a4 represents the last octet. Yet another possibility would be to represent it as a3 a2 a1 a0 with a3 = 1c, a2 = 76, a1 = 3b, and a0 = e4. In this case, a3 represents the first octet and a0 represents the last octet. No matter how this string is represented, the value of the first octet is always 1c and the value of the last octet is always e4.

5.3 Finite fields

This subclause describes the kinds of underlying finite fields $GF(q)$ that shall be used and how elements of these finite fields are to be represented for use with the primitives in 5.6. As noted in 5.2, the internal representation of objects is left to the implementation and may be different. If the internal representation is different, then conversion to the representation defined here may be needed at certain points in cryptographic operations.

5.3.1 Prime finite fields

A prime finite field is a field containing a prime number of elements. If p is a prime, then there is a unique (up to isomorphism) field $GF(p)$ with p elements, and the elements of $GF(p)$ shall be represented by the integers $0,1,2,\ldots,p-1$. A description of the arithmetic of $GF(p)$ is given in Annex A.

5.3.2 Odd characteristic extension fields

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

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

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

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

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

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

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

5.3.3 Binary finite fields

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

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

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

The polynomial basis is the set

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

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

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

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

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

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

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

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

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

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

5.3.4 Ternary finite fields

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

5.3.5 Unitary extension fields

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

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

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

5.4 Elliptic curves and points

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

5.5 Pairings

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

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

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

5.6 Data type conversion

This subclause describes the primitives that shall be used to convert between different types of objects and strings when such conversion is required in primitives, schemes, or encoding techniques. Representation of mathematical and cryptographic objects as octet strings is not specifically addressed in this subclause; rather, it is discussed in the informative Annex E. Figure 1 shows the primitives presented in this subclause and their relationships.

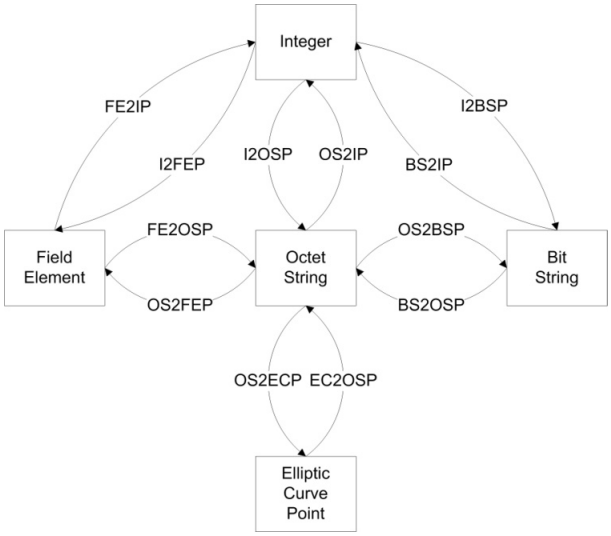

Figure 1—Data type conversion primitives

5.6.1 Converting between integers and bit strings: I2BSP and BS2IP

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

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

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

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

For example, the integer 10 945 is represented by a bit string of length 19 as 000 0010 1010 1100 0001.

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

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

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

To represent a bit string as an octet string, one simply pads enough zeroes on the left to make the number of bits a multiple of 8 and then breaks it up into octets. More precisely, a bit string $b_{l-1}b_{l-2}\ldots b_1 b_0$ of length *l* shall be converted to an octet string $M_{d-1}M_{d-2}\ldots M_1 M_0$ of length $d = \lceil l/8 \rceil$ as follows. For $0 \le i < (d-1)$, let the octet $M_i = b_{8i+7}b_{8i+6}\cdots b_{8i+1}b_{8i}$. The leftmost octet M_{d-1} shall have its leftmost $8d - l$ bits set to 0 and its rightmost $8-(8d-l)$ bits shall be $b_{l-1}b_{l-2}\ldots b_{8d-8}$.

The primitive that converts bit strings to octet strings is called the bit string to octet string conversion primitive or BS2OSP. It takes the bit string as input and outputs the octet string.

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

5.6.3 Converting between integers and octet strings: I2OSP and OS2IP

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

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

where $0 \le x_i < 256$ (note that one or more leading digits will be zero if $x < 256^{l-1}$). Then, let the octet M_i have the value x_i for $0 \le i < (l-1)$. The octet string shall be $M_{l-1}M_{l-2}\cdots M_1M_0$.

For example, the integer 10 945 is represented by an octet string of length 3 as 00 2A C1.

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

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

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

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

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

To convert an octet string back to a field element, if *q* is an odd prime or an odd prime power, then OS2IP shall be used with the octet string as the input. The primitive that converts octet strings to finite field elements is called the octet string to field element conversion primitive or OS2FEP. It takes the octet string and the field size *q* as inputs and outputs the corresponding field element. It shall output "error" if OS2BSP or OS2IP outputs "error."

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

In performing cryptographic operations, finite field elements sometimes need to be converted to non-negative integers. The primitive that performs this is called the field element to integer conversion primitive or FE2IP.

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

a) Convert the element α to an octet string using FE2OSP.

b) Convert the resulting octet string to an integer *i* using OS2IP.

c) Output *i*.

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

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

a) Convert the integer i to an octet string using I2OSP.

b) Convert the resulting octet string to a field element α using OS2FEP.

c) Output α.

5.6.6 Converting between elliptic curve points and octet strings

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

5.6.6.1 Compressed elliptic curve points

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

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

5.6.6.1.1 LSB compressed form

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

Procedures for point decompression (i.e., recovering $\hat{y}_P$ given x_P and y_P) are given in A.7.

NOTE—The term "LSB" refers to the fact that the compressed bit y_P is the least significant bit of the integer representation of $\hat{y}_P$.

5.6.6.1.2 SORT compressed form

Let (x_P, y'_P) be the inverse of the point (x_P, y_P), where $y'_P = -y_P$.

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

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

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

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

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

5.6.6.2 Two-coordinate point representations

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

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

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

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

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

— Bit S is 1 if the format uses the SORT compressed form; 0 otherwise.

— Bit U is 1 if the format is uncompressed or hybrid; 0 otherwise.

— Bit C is 1 if the format is compressed or hybrid; 0 otherwise.

— Bit $\tilde{Y}$ is equal to the bit if the format is compressed or hybrid; 0 otherwise.

— X is the octet string of length $\lceil \log_{256} q \rceil$ representing x_P according to FE2OSP (see 5.6.4).

— Y is the octet string of length $\lceil \log_{256} q \rceil$ representing y_P of P according to FE2OSP (see 5.6.4) if the format is uncompressed or hybrid; Y is an empty string if the format is compressed.

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

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

— If the first octet is not as expected for the representation

— If the octet string length is not as expected for the representation

— If an invocation of OS2FEP outputs “error”

— If an invocation of the point decompression algorithm outputs “error”

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

5.6.6.2.1 Uncompressed representation: EC2OSPXY and OS2ECPXY

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

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

5.6.6.2.2 LSB compressed representation: EC2OSPXL and OS2ECPXL

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

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

5.6.6.2.3 SORT compressed representation: EC2OSPXS and OS2ECPXS

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

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

5.6.6.2.4 LSB hybrid representation: EC2OSPXYL and OS2ECPXYL

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

In this representation, the octet *PC* shall have binary value 0000 011 Y~, where Y~ is equal to the bit in the LSB compressed form, and the octet strings *X* and *Y* shall represent x_P and y_P, respectively. The length of the octet string *PO* shall be $1+2\lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPXYL and OS2ECPXYLSORT hybrid representations: EC2OSPXYS and OS2ECPXYS.

5.6.6.2.5 SORT hybrid representation: EC2OSPXYS and OS2ECPXYS

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

In this representation, the octet *PC* shall have binary value 0000 111 Y~, where Y~ is equal to the bit 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 EC2OSPXYS and OS2ECPXYS.

5.6.6.3 *X*-coordinate-only representation: EC2OSPX and OS2ECPX

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

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

In this representation, the octet *PC* shall have binary value 0000 0001, the octet string *X* shall represent x_P, and the octet string *Y* shall be empty. The length of the octet string *PO* shall be $1+2\lceil \log_{256} q \rceil$. The corresponding primitives are called EC2OSPX and OSC2ECPX.

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

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

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

5.6.6.4 Summary of representations

Table 2 summarizes the point representations in 5.6.6.2 and 5.6.6.3.

Table 2—Elliptic curve point representations

Representation	Primitives	*PC*	*X*	*Y*	Finite fields
Uncompressed	EC2OSPXY OS2ECPXY	0000 0100	x_P	y_P	All
LSB compressed	EC2OSPXL OS2ECPXL	0000 001 Y~	x_P	Empty	$GF(p)$
SORT compressed	EC2OSPXS OS2ECPXS	0000 101 Y~	x_P	Empty	$GF(p^m)$
LSB hybrid	EC2OSPXYL OS2ECPXYL	0000 011 Y~	x_P	y_P	$GF(p)$
SORT hybrid	EC2OSPXYS OS2ECPXYS	0000 111 Y~	x_P	y_P	$GF(p^m)$
x-coordinate-only	EC2OSPX OS2ECPX	0000 0001	x_P	Empty	All
Point $\mathcal{O}$	All	0000 0000	Empty	Empty	All

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

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

6. Hashing primitives

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

6.1 Hashing to an integer

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

6.1.1 The function of IHF1

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

Input:

— A string $s \in \{0,1\}^*$

— An integer *n*

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

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

Output:

— An integer $v \in Z_n$.

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

a) If $t = 80$, then let H = SHA-1.

b) Else if $t = 112$, then let H = SHA-224.

c) Else if $t = 128$, then let H = SHA-256.

d) Else if $t = 192$, then let H = SHA-384.

e) Else if $t = 265$, then let H = SHA-512.

f) Let $v_0 = 0$.

g) Let $h_0 = 0x00\ldots00$, string of null bytes of length $2t$ bits.

h) Let $t_1 = h_0 \| s$.

i) Let $h_1 = H(t_1)$.

j) Let $a_1 = \text{OS2IP}(h_1)$.

k) Let $v_1 = a_1$.

l) Let $t_2 = h_1 \| s$.

m) Let $h_2 = H(t_2)$.

n) Let $a_2 = \text{OS2IP}(h_2)$.

o) Let $v_2 = 2^{2t} v_1 + a_2$

p) Output $v_2 \bmod n$.

6.2 Hashing to a string

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

6.2.1 Function of SHF1

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

Input:

— A string $s \in \{0,1\}^*$

— An integer n

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

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

Output:

— A string $v \in \{0,1\}^n$

Operation: Use the following steps:

a) Output $\mathrm{IHF1}(s, 2^n, t)$.

6.3 Hashing to a point in a subgroup

6.3.1 General

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

6.3.2 Function of PHF-SS

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

Input:

— A string $s \in \{0,1\}^*$

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

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

— A prime q with $q \equiv 11 (\mathrm{mod}\, 12)$ that defines the finite field $GF(q)$

— A prime p with $p \mid \#E(GF(q))$ and $p^2 \nmid \#E(GF(q))$ for the elliptic curve E defined by the flag j

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

Output:

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

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

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

b) If $j = 0$, then perform the following steps:

1) Let $y = \mathrm{IHF1}(s, q, t)$.

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

3) Let $Q = (x, y)$.

c) Else if $j = 1$, perform the following steps:

1) Let $x = \mathrm{IHF1}(s, q, t)$.

2) If the Jacobi symbol $(x/q) = +1$, then perform the following steps

i) Let $y = x^{(q+1)/4} \bmod q$.

ii) Let $Q = (x, y)$.

3) Else perform the following steps:

i) Let $y = (-x)^{(q+1)/4} \bmod q$.

ii) Let $Q = (-x, y)$.

d) Return rQ.

6.3.3 Function of PHF-GFP

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

Input:

— A string $s \in \{0,1\}^*$

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

— A prime q that defines the finite field $GF(q^e)$

— An integer e that defines the finite field $GF(q^e)$

— A prime p that defines the subgroup $E(GF(q^e))[p]$ of $E(GF(q^e))$

— An elliptic curve $E/GF(q^e): y^2 = x^3 + ax + b$

— A cofactor $r = \#E(GF(q^e))/p$

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

Output:

— An element of $E(GF(q^e))[p]$

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

a) Let $i = 0$.

b) Let x = H2FEQ(I2BSP(i)||s, t, q, e).

c) Let $t = x^3 + ax + b \in GF(q^e)$.

d) If $t = 0$, then output $(x, 0)$ and stop.

e) Apply the appropriate technique from A.2.5 to find α, the square root of t.

f) If the result of step e) indicates that no square root exists, then increment i by 1 and then go to step b).

g) Let y_1 = FE2IP(α).

h) Let y_2 = FE2IP(–α).

i) If $y_1 > y_2$ then let $y = -\alpha$ else let $y = \alpha$.

j) Let $Q = (x, y)$.

k) Return rQ.

6.3.4 Function of PHF-GF2

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

Input:

— A string $s \in \{0,1\}^*$

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

— An integer e that defines the finite field $GF(2^e)$

— A prime p that defines the subgroup $E(GF(2^e))[p]$ of $E(GF(2^e))$

— An elliptic curve $E / GF(2^e) : y^2 + xy + y = x^3 + x^2 + b$

— A cofactor $r = \#E(GF(2^e)) / p$

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

Output:

— An element of $E(GF(2^e))[p]$

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

a) Let $i = 0$.

b) Let $x = \text{H2FEQ}(i \,||\, s, t, 2, e) \in GF(2^e)$.

c) If $x = 0$, then output $(0, b^2)$ and stop.

d) Let $t = x^3 + ax^2 + b$.

e) If $t = 0$, then output $(x, 0)$ and stop.

f) Let $u = x^{-2}t$.

g) Apply the appropriate technique from A.4.7 to find an element z for which $z^2 + z = \beta$ or determine that none exist.

h) If the result of step g) indicates that no such element exists, increment i by 1 and go to step b).

i) Generate a random bit β that will be used to pick which y coordinate will be chosen for the calculated x coordinate.

j) Let $y = (z + \beta)x$.

k) Let $Q = (x, y)$.

l) Return rQ.

6.3.5 Function of PHF-GF3

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

Input:

— A string $s \in \{0,1\}^*$

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

— An integer e that defines the finite field $GF(3^e)$

— An elliptic curve $E/GF(3^e): y^2 = x^3 - x + b$ with $b = 1$ or $b = -1$

— A cofactor $r = \#E(GF(3^e)) / p$

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

Output:

— An element of $E(GF(3^e))[p]$

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

a) Let $i = 0$.

b) Let $y = \text{H2FEQ}(\text{i}||s,t,3,e) \in GF(3^e)$.

c) Let $u = y^2 - b$.

d) If $Tr(u) \neq 0$, then increment i by 1 and go to step b).

e) Use the method from A.4.10 to find an element z for which $z^3 - z = u$.

f) Generate a random element $\beta \in GF(3^e)$ that will be used to select which possible x coordinate is chosen for a particular y coordinate.

g) Let $x = z + \beta$.

h) Let $Q = (x, y)$.

i) Return rQ.

6.4 Hashing to an element of a finite field

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

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

Input:

— A string $s \in \{0,1\}^*$

— A prime q that defines the finite field $GF(q^k)$

— An integer e that defines the finite field $GF(q^k)$

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

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

Output:

— An element of $GF(q^k)$

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

a) For i = 0 to $e-1$, do the following:

1) Let $t_i = \text{IHF1}(I2BS(i) \,||\, s,q,t)$.

2) Next i.

b) Return the field element represented by $x = t_0 + t_1 x + \ldots + t_{e-1} x^{e-1}$.

7. Pairing-based primitives

7.1 General

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

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

— G_1, a group of prime order *p*.

— G_2, a group of prime order *p*.

— G_3, a group of prime order *p*.

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

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

7.2 SK primitives

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

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

The server secret is defined to be

— *s*, a random element of Z_p.

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

— G_1, G_2, G_3, e_i, the system parameters as defined in 7.1

— *Qi*: a generator of G_i

— Q_j: a generator of G_j, defined such that either $Q_j = \phi(Q_i)$ (for a supersingular curve) or $Q_j = \phi_d(Q_i)$ (for an ordinary curve)

— *R*: sQ_i

7.2.1 SK: Generation (P-SK-G)

Input:

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

— The server secret s

— An encoded identity $M \in Z_p$

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

Output:

— The derived private key K_M , an element of G_j , or "error"

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

a) If $M + s \equiv 0 (\text{mod } p)$, then output "error" and stop.

b) Compute $t = (M + s)^{-1} \bmod p$.

c) Compute $K_M = tQ_j$.

d) Output K_M.

7.2.2 SK: Verification (P-SK-V)

Input:

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

— A public-key element M in Z_p

— The corresponding private key K_M

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

Output:

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

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

a) Compute $T_1 = e_i(MQ_i + R, K_M)$.

b) Compute $T_2 = e_i(Q_i, Q_j)$.

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

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

7.2.3 SK: Encryption (P-SK-E)

Input:

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

— The public-key element M

— A message randomizer r, an integer with $0 \le r \le p - 1$

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

Output:

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

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

a) Compute $E = r(MQ_i + R)$.

b) Compute $B = e_i(Q_i, Q_j)^r$.

c) Output E and B.

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

7.2.4 SK: Decryption (P-SK-D)

Input:

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

— The public-key element M

— The associated private key K_M

— A ciphertext E

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

Output:

— A blinding factor B, an element of G_3.

Operation: Use the following steps:

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

b) Output *B*.

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

7.3 BB_1 primitives

7.3.1 General

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

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

The server secret is defined to be

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

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

— G_1, G_2, G_3, e_i: the system parameters as defined in Clause 6

— Q_i: a generator of G_i

— Q_j: a generator of G_j

— $R = s_i Q_i$

— $T = s_3 Q_i$

— $V = e_i(R, s_2 Q_j)$

7.3.2 BB_1: Generation (P-BB1-G)

Input:

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

— The server secret *s*

— A randomizer *r*, an integer in Z_p

— An encoded identity *M* in Z_p, typically derived from an identity string

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

Output:

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

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

a) Compute $t = s_1 s_2 + r(s_1 M + s_3)$ in Z_p.

b) Compute $K_{0,M} = tQ_j$.

c) Compute $K_{1,M} = rQ_j$.

d) Output $K_{0,M}$ and $K_{1,M}$.

7.3.3 BB_1: Verification (P-BB1-V)

Input:

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

— A public-key element M in Z_p

— The corresponding key $K_{0,M}$ and $K_{1,M}$

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

Output:

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

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

a) Compute $T_0 = e_i(Q_i, K_{0,M})$.

b) Compute $T_1 = e_i(MR + T, K_{1,M})$.

c) If $T_0 = T_1 V$, then output the value "valid"; otherwise, output the value "invalid."

7.3.4 BB_1: Encryption (P-BB1-E)

Input:

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

— The public-key element M in Z_p

— A message randomizer r, an element of Z_p

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

Output:

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

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

a) Compute $E_0 = rQ_i$.

b) Compute $E_1 = (rM)R + rT$.

c) Compute $B = V^r$.

d) Output E_0, E_1, and *B*.

NOTE—The point multiplications in step a) and step b) and the field exponentiation in step c) can be performed very efficiently using classic precomputation algorithms. Precomputation is viable and inexpensive because the base elements Q_i, *R*, *T*, and *V*, are all part of the public parameters and do not depend on the recipient identity.

7.3.5 BB_1: Decryption (P-BB1-D)

Input:

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

— The public-key element M

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

— A ciphertext E_o and E_1

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups and public parameters; *M* is an element of Z_p; $K_{0,M}$ and $K_{1,M}$ are elements of G_j corresponding to *M* according to P-BB1-V; and E_0 and E_1 are elements of G_i.

Output:

— A blinding factor *B*, an element of G_3

Operation: Use the following steps:

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

b) Output *B*.

NOTE—If both parties follow the algorithms specified, then the value *B* computed here is V^r, the same as that obtained in P-BB1-E (7.3.4).

7.4 BF primitives

7.4.1 General

These primitives define operations with cryptographic strength defined by reduction to the bilinear Diffie-Hellman problem. For more information on this assumption, and to review the proof that these operations reduce to this problem, see Boneh and Franklin [B28]. This system differs from the primitives earlier in this clause in that the identity is encoded into an element of G_1 instead of into Z_p. The mapping from a string to an element of G_1, which is typically an elliptic curve group, is more complex than mapping onto an integer in Z_p, which is where these primitives get their name.

For all these operations, the systems parameters have two components: a server secret and a set of public parameters.

The server secret is defined to be

— s, a random element of Z_p

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

— G_1, G_2, G_3, e, the system parameters as defined in Clause 6

— Q, a generator of G_j

— R, equal to sQ, an element of G_j

7.4.2 BF: Generation (P-BF-G)

Input:

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

— The server secret s

— An encoded identity M in G_i

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

Output:

— The derived secret D, an element of G_i

Operation: Use the following steps:

a) Compute $D = sM$.

b) Output D.

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

7.4.3 BF: Verification (P-BF-V)

Input:

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

— An encoded identity M, an element of G_i

— The purported generated key D

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; D is a BF public key and, thus, an element of G_i; M is an element G_i; and D is an element of G_i.

Output:

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

Operation: Use the following steps:

a) Compute $T_1 = e_i(D, Q)$.

b) Compute $T_2 = e_i(M, R)$.

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

7.4.4 BF: Encryption (P-BF-E)

Input:

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

— An encoded identity M, an element of G_i

— A per-message random integer r

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups.

Output:

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

Operation: Use the following steps:

a) Compute $E = rQ$.

b) Compute $B = e_i(rM, R)$.

c) Output E and B.

NOTE—The blinding factor B is the same as that obtained in P-BF-D (7.4.5).

7.4.5 BF: Decryption (P-BF-D)

Input:

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

— A ciphertext C

— A generated value D

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; C is an element of G_i; and D, an element of G_j, is a valid generated value according to P-BF-G.

Output:

— A blinding factor B, an element of G_3

Operation: Use the following steps:

a) Compute $B = e_i(C, D)$.

b) Output B.

NOTE—If both parties follow the algorithms specified, then the value B computed here is $e_i(U, X)^s$, the same blinding factor obtained in P-BF-E (7.4.4).

7.5 SCC key agreement primitives

7.5.1 Pairing-based SCC key agreement: Derive secret value (P-SCC-D1)

P-SCC-D1 is the identity-based secret value derivation primitive, SCC version. It is based on the work of Chen et al. [B42]. This primitive derives a shared secret value from a domain public key, one party's two key pairs, and another party's two public keys. If two parties correctly execute this primitive, then the parties will produce the same output. It assumes that the input keys are valid.

Input:

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

— A pairing e

— A domain public key $R \in G_i$

— The party's own identity-based private key $D \in G_j$

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

— The other party's identity-based public key $Q \in G_j$

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

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

Output:

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

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

a) Compute an elliptic curve point $A = xR$.

b) Compute a pairing $t_1 = e_i(A, Q)$.

c) Compute a pairing $t_2 = e_i(E, D)$.

d) Compute $t_3 = t_1 t_2$.

e) Compute $o_1 = \text{FE2OSP}(t_3)$.

f) Compute an elliptic curve point $B = xE$.

g) Convert *B* into an octet string *o* using EC2OSP.

h) Let $z = o \,||\, o_1$.

i) Output *z* as the shared secret value.

A conformance region should include the following:

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

— At least one valid private key *D* for each set of domain parameters

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

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

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

8. Identity-based encryption schemes

This clause describes identity-based encryption (IBE) methods and identity-based key encapsulation mechanisms (ID-KEM) for each of the family of primitives described in Clause 8. The setting is the same as the primitives, in that the schemes assume that there are three parties involved in the operations: a key server, a sender, and a recipient. Each IBE scheme is described in terms of four operations: setup, extract, encrypt, and decrypt. Each ID-KEM scheme is described with the same setup and extract operations but with encapsulate and decapsulate operations instead of with encryption and decryption.

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

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

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

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

The following operations are unique to IBE schemes:

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

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

The following operations are unique to ID-KEM schemes:

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

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

8.1 SK KEM scheme

SK-KEM uses the SK family of primitives to construct an ID-KEM mechanism that has a security reduction to the q-BSK problem. This KEM is based on the work of Sakai and Kasahara [B140] and is described in full with security proofs in Chen and Cheng [B41]. The scheme takes a security parameter *t* and a key size parameter *n*.

It requires the definition of four hash functions:

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

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

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

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

The scheme also requires a random bit generation algorithm:

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

8.1.1 SK KEM: Setup (SK-KEM-S)

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

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

a) Establish the set of base groups G_1, G_2, G_3, and a pairing $e_i : G_i \times G_j \rightarrow G_3$. Annex A contains recommendations for the generation of these objects.

b) Select a random generator Q_1 in G_i using the appropriate technique from 7.3 on an output of R_1. Calculate the corresponding generator Q_2 in G_j.

c) Generate a random server secret s in Z_p^*. Calculate the corresponding R as sQ_1.

d) Precalculate the pairing value $O = e_i(Q_1, Q_2)$.

e) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$ available. Secure the server secret s in a way appropriate to the application.

8.1.2 SK KEM: Extract (SK-KEM-EX)

The extract operation takes an arbitrary identity string *ID* in $\{0,1\}^*$ and calculates the corresponding private key K_{ID} in G_2.

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

a) Compute the identity element $M = H_1(ID)$.

b) Use the P-SK-G primitive to compute K_{ID} using M and the server secret s and public system parameters.

8.1.3 SK KEM: Encapsulate (SK-KEM-EN)

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

The steps to compute the encapsulation values are as follows:

a) Using R_1, generate a random n-bit message m.

b) Compute $r = H_3(m)$.

c) Compute P-SK-E, using r as the message randomizer.

d) Output $c = (E, B \oplus m)$.

e) Output $k = H_4(m)$.

8.1.4 SK KEM: Decapsulate (SK-KEM-DE)

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

The steps to compute the decapsulated key are as follows:

a) Parse the encapsulated value *c* as the pair (U, V).

b) Compute P-SK-D on *U*, returning the value *B*.

c) Compute $m = H_2(B) \oplus V$.

d) Compute $r = H_3(m)$.

e) Compute $Q = R + (H_1(ID) \cdot Q_1)$.

f) Verify that $U = rQ$. If not, then output "error" and stop.

g) Return $k = H_4(m)$.

8.2 BB_1 KEM scheme

BB_1-KEM uses the P-BB_1 family of primitives to construct a KEM mechanism that has a security reduction to the BDH problem. This KEM is based on the work of Boneh and Boyen [B26] and is explicitly described in Boyen [B33]. The scheme takes a security parameter *t* and a key size parameter *n*.

It requires the definition of two hash functions:

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

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

The scheme also requires a random bit generation algorithm:

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

8.2.1 BB_1 KEM: Setup (BB1-KEM-S)

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

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

a) Establish the set of base groups G_1, G_2, G_3 and a pairing $e_i : G_i \times G_j \rightarrow G_3$. Annex A contains recommendations for the generation of these objects.

b) Select a random generator Q_1 in G_i using the appropriate technique from 7.3 on an output of R_1.

c) Select a random generator Q_2 in G_j that may, but does need not to be, related to Q_1 by an explicit mapping ϕ.

d) Generate random server secrets s_1, s_2, and s_3 in Z_p.

e) Calculate the corresponding *R* as $s_1 Q_1$ and *T* as $s_3 Q_1$.

f) Calculate the pairing value *V* as $e_i(s_1 Q_1, s_2 Q_2)$.

g) Make the public parameter set $\mathcal{P} = (Q_1, Q_2, R, T, V, G_1, G_2, e_i)$ available. Secure the server secrets s_1, s_2, and s_3 in a way appropriate to the application.

8.2.2 BB_1 KEM: Extract (BB1-KEM-EX)

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

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

a) Compute the identity element $M = H_1(ID)$.

b) Use the P-BB1-G primitive to compute $K_{0,ID}$ and $K_{1,ID}$ using *M*, the server secrets s_1, s_2, and s_3, the public system parameters.

8.2.3 BB_1 KEM: Encapsulate (BB1-KEM-EN)

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

The steps to compute the encapsulation values are as follows:

a) Using R_1, generate a random integer *r*.

b) Compute P-BB1-E, using *r* as the message randomizer.

c) Output c as the pair (E_0, E_1).

d) Output $k = H_2(B)$.

8.2.4 BB_1 KEM: Decapsulate (BB1-KEM-DE)

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

The steps to compute the decapsulated key are as follows:

a) Parse the encapsulated value *c* as the pair (E_0, E_1).

b) Compute PBB1D on (E_0, E_1) to obtain the value *B*.

c) Return $k = H_2(B)$.

8.3 BB_1 IBE scheme

BB_1 IBE uses the P-BB1 family of primitives to construct an IBE mechanism that has a security reduction to the BDH problem. This cryptosystem is based on the work of Boneh and Boyen [B26]. The scheme takes a security parameter *t* and a key size parameter *n*.

It requires the definition of three hash functions:

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

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

— $H_3 : G_3 \times \{0,1\}^n \times G_1 \times G_1 \rightarrow Z_p$, where

$$H_3(x, s, P_1, P_2) = \text{IHF1}(\text{FE2OSP}(x) \,||\, \text{BS2OSP}(s) \,||\, \text{EC2OSP}(P_1) \,||\, \text{EC2OSP}(P_2), p, t)$$

The scheme also requires a random bit generation algorithm:

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

8.3.1 BB_1 IBE: Setup (BB1-IBE-S)

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

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

a) Establish the set of base groups G_1, G_2, G_3 and a pairing $e_i : G_i \times G_j \rightarrow G_3$. Annex A contains recommendations for the generation of these objects.

b) Select a random generator Q_1 in G_i using the appropriate technique from 7.3.1 on an output of R_1.

c) Select also a random generator Q_2 in G_j, which may but need not be related to Q_1 by an explicit mapping ϕ.

d) Generate random server secrets s_1, s_2, and s_3, in Z_p.

e) Calculate the corresponding *R* as $s_1 Q_1$ and *T* as $s_3 Q_1$.

f) Calculate the pairing value *V* as $e_i(s_1 Q_1, s_2 Q_2)$.

g) Make the public parameter set $\mathcal{P} = (Q_1, Q_2, R, T, V, G_1, G_2, e_i)$ available. Secure the server secrets s_1, s_2, and s_3 in a way appropriate to the application.

8.3.2 BB$_1$ IBE: Extract (BB1-IBE-EX)

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

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

a) Compute the identity element $M = H_1(ID)$.

b) Use the P-BB1-G primitive to compute $K_{0,ID}$ and $K_{1,ID}$ using *M*, the server secrets $K_{0,ID}$ and $K_{1,ID}$ using *M*, the server secrets s_1, s_2, s_3, and the public system parameters.

8.3.3 BB$_1$ IBE: Encrypt (BB1-IBE-EN)

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

The steps to compute the ciphertext are as follows:

a) Using R_1, generate a random integer *r* in Z_p.

b) Compute P-BB1-E, using *r* as the message randomizer.

c) Compute $Y = H_2(B) \oplus M$.

d) Compute $t = r + H_3(B, Y, E_0, E_1)$ in Z_p.

e) Output the quadruple $c = (Y, E_0, E_1, t)$ as the ciphertext.

8.3.4 BB$_1$ IBE: Decrypt (BB1-IBE-DE)

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

The steps to compute the decapsulated key are as follows:

a) Parse *c* as the quadruple $c = (Y, E_0, E_1, t)$.

b) Compute P-BB1-D on (E_0, E_1), to obtain the value *B*.

c) Compute $r = t - H_3(B, Y, E_0, E_1)$ in Z_p.

d) Verify that $B = V^r$ and that $E_0 = rQ_1$. If not, then output "error" and stop.

e) Return $M = Y \oplus H_2(B)$.

NOTE—The verifications in step d) can be performed very efficiently using classic precomputation methods for field exponentiation and curve multiplication. Precomputation is viable because the base elements V and Q_1 are part of the public parameters and thus constant and independent of the recipient identity.

8.4 BF IBE scheme

BF IBE uses the P-BF family of primitives to construct an IBE scheme that has a security reduction to the BDH problem. This IBE scheme is based on the work of Boneh and Franklin [B28] and is described with full security proofs in that work. The scheme takes a security parameter *t* and a key size parameter *n*. The following scheme uses the Fujisaki-Okamoto transform to create an IND-CCA secure IBE scheme.

BF IBE requires the definition of four hash functions:

— $H_1 : \{0,1\}^* \rightarrow G_1$, where H_1 is defined to be one of PHF-SS, PHF-GFP, and PHF-GF2, the choice of which will depend on the elliptic curve chosen to define G_1

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

— $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow Z_p^*$, where $H_3(s_1, s_2) = \mathrm{IHF1}(\mathrm{BS2OSP}(s_1) \,||\, \mathrm{BS2OSP}(s_2)), p-1, t)$

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

The scheme also requires a random bit generation algorithm:

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

8.4.1 BF IBE: Setup (BF-IBE-S)

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

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

a) Establish the set of base groups G_1, G_2, and G_3, and a pairing $e_i : G_i \times G_j \rightarrow G_3$. Annex A contains recommendations for the generation of these objects. G_i shall be of the form $E(GF(q))[p]$, where *p* and *q* are primes.

b) Select a random generator Q in G_j using A.7.15.

c) Generate a random server secret s in Z_p^*.

d) Calculate the corresponding *R* as sQ.

e) Make the public parameter set $P = (R, Q, G_1, G_2, G_3, e_i)$ available. Secure the server secret *s* in a way appropriate to the application.

8.4.2 BF IBE: Extract (BF-IBE-EX)

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

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

a) Compute the identity element $M = H_1(ID)$.

b) Use the P-BF-G primitive to compute K_{ID} using *M* and the server secret *s* and public system parameters.

8.4.3 BF IBE: Encrypt (BF-IBE-EN)

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

The steps to compute the ciphertext *E* are as follows:

a) Using R_1, compute an *n*-bit message randomizer *o*.

b) Compute $M = H_1(ID)$.

c) Compute $r = H_3(o, M)$.

d) Compute $C_1 = rQ$.

e) Compute the blinding value *B* using P-BF-EN with *M*, *r* and the public parameters $\mathcal{P}$.

f) Compute $C_2 = o \oplus H_2(B)$.

g) Compute $C_3 = m \oplus H_4(o)$.

h) Output the ciphertext $E = (C_1, C_2, C_3)$.

8.4.4 BF IBE: Decrypt (BF-IBE-DE)

The decrypt operation takes a ciphertext *E* and the private key K_{ID} from the BF-IBE-EX operation and either recovers the message *m* or outputs "error."

The steps to decrypt *E* into *m* are as follows:

a) Using P-BF-D with C_1 as the ciphertext, and K_{ID} as the key, compute the blinding value *B*.

b) Compute $o = C_2 \oplus H_2(B)$.

c) Compute $m = C_3 \oplus H_4(o)$.

d) Compute $r = H_3(o, M)$.

e) Test if $C_1 = rQ$. If not, then output "error" and stop.

f) Output *m* as the decrypted message.

9. Identity-based signature schemes

9.1 BLMQ signature scheme

9.1.1 General

The BLMQ signature uses the DHI family of primitives to construct an identity-based signature that has a security reduction to an assumption that is related to (and actually weaker than) the *q*-BDHI assumption. This signature is a noninteractive proof of knowledge of a private key generated according to the work of Sakai and Kasahara [B140].

It requires the definition of the following two hash functions:

— $H_1 : \{0,1\}^* \rightarrow Z_p$

— $H_2 : \{0,1\}^* \rightarrow Z_q$

9.1.2 BLMQ signature: Setup (BLMQ-SIG-S)

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

Input:

— A security parameter *t*

Output:

— A public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$ and a server secret *s*

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

a) Establish the set of base groups G_1, G_2, G_3, and a pairing $e_i : G_i \times G_j \rightarrow G_3$ as described in Annex A.

b) Pick a random generator Q_2 in G_j.

c) Calculate $Q_1 = \phi(Q_2)$ in G_i if ϕ is defined.

d) Generate a random server secret *s* in Z_p^*.

e) Calculate $R = sQ_2$.

f) Calculate $O = e_i(Q_1, Q_2)$.

g) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$ available.

9.1.3 BLMQ signature: Extract (BLMQ-SIG-EX)

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

Input:

— A set of public parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$

— A server secret *s*

— An identity *ID* in $\{0,1\}^*$

Output:

— A private key K_{ID}

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

a) Compute the identity element $M = H_1(ID)$ in Z_p.

b) Compute $K_{ID} = (M + s)^{-1} Q_1$ using the server secret *s*.

9.1.4 BLMQ signature: Create signature (BLMQ-SIG-SI)

Input:

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

— The public-key element *M*

— The associated private key $K_{ID} = (M + s)^{-1} Q_1$

— A randomizer *r*, an integer

— A message *m* to be signed

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

Output:

— A signature (h, S), where *h* is an element of Z_p and *S* is an element of G_i.

Operation: Use the following steps:

a) Compute $u = e_i(Q_1, Q_2)^r$.

b) Compute $h = H_2(m, u)$.

c) Compute $S = (r + h)K_m$.

d) Output *h* and *S*.

9.1.5 BLMQ signature: Verify signature (BLMQ-SIG-VE)

Input:

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

— The public-key element *M*

— A signature (*h*, *S*)

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; *R* is a public key, an element of G_j; *M* is an element of Z_p; K_M, an element of G_i, is a valid generated value for *M*; *h* is an element of Z_p; and *S* is an element of G_i.

Output:

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

Operation: Use the following steps:

a) Compute $u = \dfrac{e_i(S, MQ_2 + R)}{e_i(Q_1, Q_2)^h}$.

b) Output "valid" if $h = H_2(m, u)$ and "invalid" otherwise.

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

10. Identity-based signcryption schemes

10.1 BLMQ signcryption scheme

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

It requires the definition of the following three hash functions:

— $H_1 : \{0,1\}^* \rightarrow Z_p$

— $H_2 : \{0,1\}^* \rightarrow Z_q$

— $H_3 : G_3 \rightarrow \{0,1\}^n$

10.1.1 BLMQ signcryption: Setup (BLMQ-SC-S)

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

The steps to set up the key server and system-wide parameters $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$ are as follows:

a) Establish the set of base groups G_1, G_2, G_3, and a pairing $e_i : G_i \times G_j \rightarrow G_3$.

b) Pick a random generator Q_2 in G_j. Calculate the corresponding $Q_1 = \phi(Q_2)$ in G_i.

c) Generate a random server secret s in Z_p^*. Calculate the corresponding R as sQ_2.

d) Precalculate the pairing value $O = e_i(Q_1, Q_2)$.

e) Make the public parameter set $\mathcal{P} = (R, O, Q_1, Q_2, G_1, G_2, G_3, e_i)$ available.

10.1.2 BLMQ signcryption: Extract (BLMQ-SC-EX)

The extract operation takes an arbitrary identity string *ID* in $\{0,1\}^*$ and calculates the corresponding private key K_{ID} in G_j. Users' keys shall lie in G_j.

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

a) Compute the identity element $M = H_1(ID)$ in Z_p.

b) Compute $K_{ID} = (M + s)^{-1} Q_2$ using the server secret s.

10.1.3 BLMQ signcryption: Sign and encrypt (BLMQ-SC-SE)

Input:

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

— The sender's public identifier M_A

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

— The receiver's identifier M_B

— A randomizer r, an integer with $0 \leq r \leq p - 1$

— A message m to be jointly signed and encrypted

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

Output:

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

Operation: Conduct the following steps:

a) Compute $u = e_i(Q_1, Q_2)^r$.

b) Compute $h = H_2(m, u)$.

c) Compute $S = (r + h)K_A$.

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

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

f) Output the ciphertext (c, S, T).

10.1.4 BLMQ signcryption: Decrypt and verify (BLMQ-SC-DV)

Input:

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

— The sender's public identifier M_A

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

— A ciphertext (c, S, T)

Input constraints: The parameters $\mathcal{P}$ describe valid bilinear groups; R is a public key, an element of G_j; M_A and M_B are both elements of Z_p; K_B, an element of G_j, is a valid generated value for M_B; h is an element of Z_p; and S is an element of G_i.

Output:

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

Operation: Use the following steps:

a) Compute $u = e_i(T, K_B)$.

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

c) Compute $h = H_2(m, u)$.

d) Accept the message if $u = \dfrac{e_i(S, M_A Q_2 + R)}{e_i(Q_1, Q_2)^h}$.

e) If the test of step d) succeeds, output the plaintext m and the signature (h, S).

11. Identity-based key agreement schemes

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

A key agreement scheme consists of a key agreement operation, along with supporting key management. Domain parameter and key pair generation for the key agreement schemes are specified in Clause 11. A key agreement operation has the following form for all the schemes:

a) An administrator establishes one or more sets of valid domain parameters with which the parties' key pairs shall be associated.

b) Users obtain their valid private keys associated with the domain parameters established in step a).

c) Users obtain the other party's identity-based public key and one or more purported public keys for the operation.

d) (Optional) Depending on the cryptographic operations in step c), users choose an appropriate method to validate the public keys and the domain parameters. If any validation fails, output "invalid" and stop.

e) Users apply certain cryptographic operations to the private and public keys to produce a shared secret value.

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

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

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

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

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

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

— Private key not found in step b)

— Public key not found in step d)

— Public key not valid in step e)

— Private key or public key not supported in step f)

— Key derivation parameter not supported in step f)

NOTE 6—Such error conditions should be detected and handled appropriately by an implementation, but the specific methods for detecting and handling them are outside of the scope of this standard.

11.1 Wang key agreement scheme

The Wang key agreement scheme is an IBKAS-IDAK, identity-based key agreement scheme, IDAK version. In this scheme, each party contributes two key pairs.

The following scheme options shall be established between the parties to the scheme:

— A secret value derivation operation, which shall be P-WKA-D1

— A key derivation function, which should be KDF1 as defined in subclause 13.1 of IEEE Std 1363-2000, or a function designated for use with IBKAS-IDAK in an addendum to this standard

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

These primitives define operations with cryptographic strength defined by reduction to the bilinear Diffie-Hellman problem (see 7.3).

For these operations, the system parameters comprise a server secret and a set of public parameters.

The server secret is defined to be

— s, a random element of Z_p

The pubic parameters are defined to be $P = (G_1,G_2,G_3,e_i,Q,R,H)$, where

— G_1, G_2, G_3, e_i are as defined in Clause 7

— Q is a generator of G_i

— R, equal to sQ, an element of G_i

— H is a cryptographic hash function H that maps an octet string to an octet string of length $h = \lceil \log_2 |G_1| / 2 \rceil$

11.1.1 Wang key agreement: Derive public key (WKA-KA-D1)

This operation derives an entity's public key from its public identity string.

Input:

— The public parameters $P = (G_1,G_2,G_3,e_i,Q,R,H)$

— Parameters t, j, q, p, where t is a security parameter, j is the flag defining the supersingular elliptic curve as specified in A.11, and q and p are system parameters

— An octet string *ID* representing a user's identity

Output:

— The derived public key W, which is a generator of G_j ; or "error"

Operation:

a) Compute $W = \text{PHF-SS}(ID, t, j, q, p)$.

b) Output W.

11.1.2 Wang key agreement: Derive private key (WKA-KA-D2)

This operation derives an entity's private key from a public key and a server secret.

Input:

— The public parameters $P = (G_1, G_2, G_3, e_i, Q, R, H)$

— The server secret s

— The public key W

Input constraints: The parameters $\mathcal{P}$, the server secret s, and the public key W are valid.

Output:

— The derived private key U, which is an element of G_j , or "error."

Operation: Use the following steps:

a) Compute $U = sW$ using P-BF-G.

b) Output U.

11.1.3 Wang key agreement: Verification (WKA-KA-V)

This operation verifies that a private key is valid.

Input:

— The public parameters $P = (G1, G2, G3, ei, Q, R, H)$

— A public key W, an element of G_j

— A private key U, an element of G_j

Input constraints: The public parameters, public key, and private key are valid.

Output:

— The value "valid" if U is the private key corresponding to W and "invalid" otherwise.

Operation: Perform the following steps:

a) Compute $T_1 = e_i(Q, U)$ using P-BF-D.

b) Compute $T_2 = e_i(R,W)$ using P-BF-D.

c) If $T_1 = T_2$, then output "valid"; otherwise output "invalid."

11.1.4 Wang key agreement: Derive secret value (WKA-KA-D3)

This operation derives a secret shared value from one party's two key pairs and another party's two public keys. If two parties correctly execute this primitive, then the two parties will produce the same output. This operation can be invoked by a scheme to derive a shared secret key; specifically, it may be used with this scheme.

Input:

— The public parameters $P = (G_1,G_2,G_3,e_i,Q,R,H)$

— The first party's identity-based private key U, an element of G_j

— The first party's second key pair (u,V), where u is an element of Z_p and $V = uW$

— The second party's identity-based public key W'

— The second party's second public key V'

— A cryptographic hash function H that maps an octet string to an octet string of length h

— A flag f that takes the values 0 and 1. The value 0 indicates the role of initiator; the value 1 indicates the role responder

Input constraints: The public parameters $\mathcal{P}$ are valid; the keys $U, (u,V), W', V'$ are valid and are all associated with the parameters $\mathcal{P}$.

Output:

— The derived shared secret value z, an element of $GF(q)^*$; or the message "error"

Operation: Perform the following steps:

a) Let $o = \text{EC2OSPX}(V)$.

b) Let $o' = \text{EC2OSPX}(V')$.

c) Compute an octet string $t = H(o \| o')$.

d) Convert t into an integer using OS2IP.

e) Compute an octet string $t' = H(o' \| o)$.

f) Convert t' into an integer using OS2IP.

g) Compute $P = (u + i)U$.

h) Compute $R = i'W' + V'$.

i) If $f = 0$, then let $z = e_i(P,R)$.

j) If $f = 0$, then let $z = e_i(R,P)$.

k) If $z = 1$, then terminate.

l) Output *z*.

11.1.5 Wang key agreement: Generate shared secrets (WKA-KA-G)

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

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

b) Obtain the identity-based private key *w* and select a valid key pair (u,v) for the operation, associated with the parameters established in step a).

c) Obtain the other party's identity-based public key w' and the purported public key v' for the operation, associated with the parameters established in step a).

d) Compute a shared secret value *z* from the selected private keys *w* and *u* and the other party's two public keys w' and v' with the selected secret value derivation primitive.

e) Convert the shared secret value *z* to an octet string *Z* using FE2OSP.

f) For each shared secret key to be agreed on

   1) Establish or otherwise agree on key derivation parameters P_i for the key.

   2) Derive a shared secret key K_i from the octet string *Z* and the key derivation parameters P_i with the selected key derivation function (see 9.1.5).

A conformance region should include the following:

— At least one valid set of domain parameters

— At least one valid private key *w* for each set of domain parameters

— All valid key pairs (u,v) associated with the same set of domain parameters as *w*

— All valid public keys w' and v' associated with the same set of domain parameters as *s*; if key validation is performed, then the invalid public keys w' and v' that are appropriately handled by the implementation may also be included in the conformance region

— A range of key derivation parameters *P*

11.2 SCC key agreement scheme

The security of this scheme was described by Chen et al. [B42]. Each party contributes two key pairs.

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

— A secret value derivation primitive, which shall be P-SCC-D1

— A key derivation function, which should be KDF1 (subclause 13.1 of IEEE Std 1363-2000) or a function designated for use with SCC key agreement in an addendum to this standard

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

11.2.1 SCC key agreement: Generate shared secrets (SCC-KA-G)

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

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

b) Obtain the identity-based private key D and select a valid key pair $(x, E = xP)$ using P-BF-G, with the parameters established in step a).

c) Obtain the other party's identity-based public key Q and the purported public key E' for the operation, associated with the parameters established in step a).

d) Compute a shared secret value z from the selected private keys D and x and the other party's two public keys Q and E with the selected secret value derivation primitive P-SCC-D1.

e) Convert the shared secret value z to an octet string Z using FE2OSP.

f) For each shared secret key to be agreed on

   1) Establish or otherwise agree on key derivation parameters P_i for the key.

   2) Derive a shared secret key K_i from the octet string Z and the key derivation parameters P_i with the selected key derivation function (see 9.1.4).

A conformance region should include the following:

— At least one valid set of domain parameters

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

— All valid key pairs (x, E) associated with the same set of domain parameters as s

— All valid public keys Q and P_i associated with the same set of domain parameters as s; if key validation is performed, then the invalid public keys Q and P_i that are appropriately handled by the implementation may also be included in the conformance region

— A range of key derivation parameters P

Annex A

(informative)

Number-theoretic background

A.1 Integer and modular arithmetic: Overview

A.1.1 Modular arithmetic

A.1.1.1 Modular reduction

Modular arithmetic is based on a fixed integer $m > 1$ called the modulus. The fundamental operation is reduction modulo m. To reduce an integer a modulo m, one divides a by m and takes the remainder r. This operation is written as follows:

$r := a \bmod m$.

The remainder shall satisfy $0 \leq r < m$.

Examples:

$11 \bmod 8 = 3$

$7 \bmod 9 = 7$

$-2 \bmod 11 = 9$

$12 \bmod 12 = 0$

Congruences

Two integers a and b are said to be congruent modulo m if the two integers have the same result with reduction modulo m. This relationship is written as follows:

$a \equiv b \pmod m$

Two integers are congruent modulo m if and only if their difference is divisible by m.

Example:

$11 \equiv 19 \pmod 8$

If $r = a \bmod m$, then $r \equiv a \pmod m$.

If $a_0 \equiv b_0 \pmod{m}$ and $a_1 \equiv b_1 \pmod{m}$, then

$$a_0 + a_1 \equiv b_0 + b_1 \pmod{m}$$

$$a_0 - a_1 \equiv b_0 - b_1 \pmod{m}$$

$$a_0 a_1 \equiv b_0 b_1 \pmod{m}$$

A.1.1.2 Integers modulo *m*

The integers modulo m are the possible results of reduction modulo m. Thus, the set of integers modulo m is

$$Z_m = \{0, 1, \ldots, m - 1\}$$

One performs addition, subtraction, and multiplication on the set Z_m by performing the corresponding integer operation and reducing the result modulo m. For example, in Z_7

$$3 = 6 + 4$$

$$5 = 1 - 3$$

$$6 = 4 \times 5$$

A.1.1.3 Modular exponentiation

If v is a positive integer and g is an integer modulo m, then modular exponentiation is the operation of computing $g^v \bmod m$ (also written $\exp(g, v) \bmod m$). Subclause A.2.1 contains an efficient method for modular exponentiation.

A.1.1.4 GCDs and LCMs

If m and h are integers, the greatest common divisor (or GCD) is the largest positive integer d dividing both m and h. If $d = 1$, then m and h are said to be relatively prime (or coprime). Subclause A.2.2 contains an efficient method for computing the GCD.

The least common multiple (or LCM) is the smallest positive integer l divisible by both m and h. The GCD and LCM are related by

$$\text{GCD}(h, m) \times \text{LCM}(h, m) = hm$$

(for h and m positive), so that the LCM is easily computed if the GCD is known.

A.1.1.5 Modular division

The multiplicative inverse of h modulo m is the integer k modulo m such that $hk \equiv 1 \pmod{m}$. The multiplicative inverse of h is commonly written as $h^{-1} \pmod{m}$. It exists if h is relatively prime to m and not otherwise.

If g and h are integers modulo m, and h is relatively prime to m, then the *modular quotient* g/h modulo m is the integer $gh^{-1} \bmod m$. If c is the modular quotient, then c satisfies $g \equiv hc \pmod{m}$.

The process of finding the modular quotient is called modular division. Subclause A.2.2 contains an efficient method for modular division.

A.1.2 Prime finite fields

A.1.2.1 Field *GF*(*p*)

In the case in which m equals a prime p, the set Z_p forms a prime finite field and is denoted $GF(p)$.

In the finite field $GF(p)$, modular division is possible for any denominator other than 0. The set of nonzero elements of $GF(p)$ is denoted $GF(p)^*$.

A.1.2.2 Orders

The order of an element c of $GF(p)^*$ is the smallest positive integer v such that $c^v \equiv 1 \pmod{p}$. The order always exists and divides $p - 1$. If k and l are integers, then $c^k \equiv c^l \pmod{p}$ if and only if $k \equiv l \pmod{v}$.

A.1.2.3 Generators

If v divides $p - 1$, then there exists an element of $GF(p)$* having order v. In particular, there always exists an element g of order $p - 1$ in $GF(p)$*. Such an element is called a generator for $GF(p)$* because every element of $GF(p)$* is some power of g. In number-theoretic language, g is also called a primitive root for p.

A.1.2.4 Exponentiation and discrete logarithms

Suppose that the element g of $GF(p)^*$ has order v. Then, an element h of $GF(p)^*$ satisfies

$$h \equiv g^l \pmod{p}$$

for some l if and only if $h^v \equiv 1 \pmod{p}$. The exponent l is called the discrete logarithm of h (with respect to the base g). The discrete logarithm is an integer modulo v.

A.1.3 Modular square roots

A.1.3.1 Legendre symbol

If $p > 2$ is prime and a is any integer, then the Legendre symbol (a / p) is defined as follows. If p divides a, then $(a / p) = 0$. If p does not divide a, then (a / p) equals 1 if a is a square modulo p and –1 otherwise. (Despite the similarity in notation, a Legendre symbol should not be confused with a rational fraction; the distinction shall be made from the context.)

Algorithms for computing Legendre symbol are given in A.2.3.

A.1.3.2 Square roots modulo a prime

Let p be an odd prime and let g be an integer with $0 \le g < p$. A square root modulo p of g is an integer z with $0 \le z < p$ and

$$z^2 \equiv g \pmod{p}$$

The number of square roots modulo p of g is $1+J$, where J is the Jacobi symbol (g / p).

If $g = 0$, then there is one square root modulo p, namely, $z = 0$. If $g \neq 0$, then g has either 0 or 2 square roots modulo p. If z is one square root, then the other is $p – z$.

A procedure for computing square roots modulo a prime is given in A.2.5.

A.2 Integer and modular arithmetic: Algorithms

A.2.1 Modular exponentiation

Modular exponentiation can be performed efficiently by the binary method outlined in the following.

Input: a positive integer v, a modulus m, and an integer g modulo m.

Output: $g^v \bmod m$.

1. Let $v = v_r v_{r-1} \ldots v_1 v_0$ be the binary representation of v, where the most significant bit v_r of v is 1.
2. Set $x \leftarrow g$.
3. For i from $r – 1$ down to 0 do
   3.1 Set $x \leftarrow x^2 \bmod m$.
   3.2 If $v_i = 1$, then set $x \leftarrow gx \bmod m$.
4. Output x.

There are several modifications that improve the performance of this algorithm**.** These methods are summarized in Gordon [B68].

A.2.2 Extended Euclidean algorithm

The following algorithm computes efficiently the GCD d of m and h. If m and h are relatively prime, then the algorithm also finds the quotient g/h modulo m.

Input: an integer $m > 1$ and integers g and $h > 0$. (If only the GCD of m and h is desired, then no input g is required.)

Output: the GCD d of m and h, and if $d = 1$, then the integer c with $0 < c < m$ and $c \equiv g/h \pmod{m}$.

1. If $h = 1$, then output $d := 1$ and $c := g$ and stop.
2. Set $r_0 \leftarrow m$.
3. Set $r_1 \leftarrow h \bmod m$.
4. Set $s_0 \leftarrow 0$.
5. Set $s_1 \leftarrow g \bmod m$.
6. While $r_1 > 0$
   - 6.1 Set $q \leftarrow \lfloor r_0 / r_1 \rfloor$.
   - 6.2 Set $r_2 \leftarrow r_0 - qr_1 \bmod m$.
   - 6.3 Set $s_2 \leftarrow s_0 - qs_1 \bmod m$.
   - 6.4 Set $r_0 \leftarrow r_1$.
   - Set $r_1 \leftarrow r_2$.
   - Set $s_0 \leftarrow s_1$.
   - Set $s_1 \leftarrow s_2$.
7. Output $d := r_0$.
8. If $r_0 = 1$, then output $c := s_0$.

If m is prime, then the quotient g/h exists provided that $h \not\equiv 0 \pmod{m}$ and can be found efficiently using exponentiation via

$$c := g\, h^{m-2} \bmod m$$

A.2.3 Evaluating Legendre symbols

The following algorithm efficiently computes the Legendre symbol.

Input: an integer a and a prime $p > 2$.

Output: the Legendre symbol (a / p).

1. Set $x \leftarrow a$, $y \leftarrow p$, $L \leftarrow 1$.
2. While $y > 1$
   - 2.1 Set $x \leftarrow (x \bmod y)$.
   - 2.2 If $x > y/2$, then
     - 2.2.1 Set $x \leftarrow y - x$.
     - 2.2.2 If $y \equiv 3 \pmod 4$, then set $L \leftarrow -L$.
   - 2.3 If $x = 0$, then set $x \leftarrow 1$, $y \leftarrow 0$, $L \leftarrow 0$.
   - 2.4 While 4 divides x
     - 2.4.1 Set $x \leftarrow x/4$.
   - 2.5 If 2 divides x, then
     - 2.5.1 Set $x \leftarrow x/2$.
     - 2.5.2 If $y \equiv \pm 3 \pmod 8$, then set $L \leftarrow -L$.
   - 2.6 If $x \equiv 3 \pmod 4$ and $y \equiv 3 \pmod 4$, then set $L \leftarrow -L$.

2.7 Switch x and y.

3. Output L.

The Legendre symbol can also be found efficiently using exponentiation via

$$\left(\frac{a}{p}\right) := a^{(p-1)/2} \bmod p$$

A.2.4 Generating Lucas sequences

Let P and Q be nonzero integers. The Lucas sequence V_k for P, Q is defined by

$$V_0 = 2,\ V_1 = P, \text{ and } V_k = PV_{k-1} - QV_{k-2} \text{ for } k \geq 2$$

This recursion is adequate for computing V_k for small values of k. For large k, one can compute V_k modulo an odd integer $n > 2$ using the following algorithm (see Joye and Quisquater [B90]). The algorithm also computes the quantity $Q^{\lfloor k/2 \rfloor} \bmod n$; this quantity will be useful in the application given in A.2.5.

Input: an odd integer $n > 2$, integers P and Q, and a positive integer k.

Output: $V_k \bmod n$ and $Q^{\lfloor k/2 \rfloor} \bmod n$.

1. Set $v_0 \leftarrow 2$, $v_1 \leftarrow P$, $q_0 \leftarrow 1$, $q_1 \leftarrow 1$.
2. Let $k = k_r k_{r-1} \ldots k_1 k_0$ be the binary representation of k, where the leftmost bit k_r of k is 1.
3. For i from r downto 0, do the following:
   3.1 Set $q_0 \leftarrow q_0\, q_1 \bmod n$.
   3.2 If $k_i = 1$, then set
   $q_1 \leftarrow q_0\, Q \bmod n$.
   $v_0 \leftarrow v_0\, v_1 - P\, q_0 \bmod n$.
   $v_1 \leftarrow v_1^2 - 2\, q_1 \bmod n$.
   else set
   $q_1 \leftarrow q_0$.
   $v_1 \leftarrow v_0\, v_1 - P\, q_0 \bmod n$.
   $v_0 \leftarrow v_0^2 - 2\, q_0 \bmod n$.
4. Output v_0 and q_0.

A.2.5 Finding square roots modulo a prime

The following algorithm computes a square root z modulo p of $g \neq 0$.

Input: an odd prime p, and an integer g with $0 < g < p$.

Output: a square root modulo p of g if one exists. (In case III, the message "no square roots exist" is returned if none exists.)

I. $p \equiv 3 \pmod 4$, that is $p = 4k + 3$ for some positive integer k. (See Lehmer [B104].)
   1. Compute and output $z := g^{k+1} \bmod p$.

II. $p \equiv 5 \pmod 8$, that is $p = 8k + 5$ for some positive integer k. (See Atkin [B16].)
   1. Compute $\gamma := (2g)^k \bmod p$.

2. Compute $i := 2g\gamma^2 \bmod p$.
3. Compute and output $z := g\gamma(i - 1) \bmod p$.

III $p \equiv 1 \pmod 8$. (See Lehmer [B104].)

1. Set $Q \leftarrow g$.
2. Generate a value P with $0 < P < p$ not already chosen.
3. Compute via A.2.4 the quantities

$$V := V_{(p+1)/2} \bmod p \text{ and } Q_0 := Q^{(p-1)/4} \bmod p$$

4. Set $z \leftarrow V / 2 \bmod p$.
5. If $(z^2 \bmod p) = g$, then output z and stop.
6. If $1 < Q_0 < p - 1$, then output the message "no square roots exist" and stop.
7. Go to step 2.

NOTE 1—To perform the modular division of an integer V by 2 (needed in step 4 of case III), one can simply divide by 2 the integer V or $V + p$ (whichever is even). (The integer division by 2 can be accomplished by shifting the binary expansion of the dividend by one bit.)

NOTE 2—As written, the algorithm for case III works for all $p \equiv 1 \pmod 4$, although it is less efficient than the algorithm for case II when $p \equiv 5 \pmod 8$.

NOTE 3—In case III, a given choice of P will produce a solution if and only if $P^2 - 4Q$ is not a quadratic residue modulo p. If P is chosen at random, then the probability of this is at least 1/2. Thus, only a few values of P will be required. It may therefore be possible to speed up the process by restricting to very small values of P and implementing the multiplications by P in A.2.4 by repeated addition.

NOTE 4—In cases I and II, the algorithm produces a solution z provided that one exists. If it is unknown whether a solution exists, then the output z should be checked by comparing $w := z^2 \bmod p$ with g. If $w = g$, then z is a solution; otherwise, no solutions exist. In case III, the algorithm performs the determination of whether a solution exists.

A.2.6 Finding square roots modulo a power of 2

If $r > 2$ and $a < 2^r$ is a positive integer congruent to 1 modulo 8, then there is a unique positive integer b less than 2^{r-2} such that $b^2 \equiv a \pmod{2^r}$. The number b can be computed efficiently using the following algorithm. The binary representations of the integers *a, b, h* are denoted as

$$a = a_{r-1}...a_1a_0$$

$$b = b_{r-1}...b_1b_0$$

$$h = h_{r-1}...h_1h_0$$

Input: an integer $r > 2$, and a positive integer $a \equiv 1 \pmod 8$ less than 2^r.

Output: the positive integer b less than 2^{r-2} such that $b^2 \equiv a \pmod{2^r}$.

1. Set $h \leftarrow 1$.
2. Set $b \leftarrow 1$.
3. For j from 2 to $r - 2$, do the following:
   If $h_{j+1} \neq a_{j+1}$, then
      Set $b_j \leftarrow 1$.
      If $2j < r$

then $h \leftarrow (h + 2^{j+1}b - 2^{2j}) \bmod 2^r$.
else $h \leftarrow (h + 2^{j+1}b) \bmod 2^r$.

4. If $b_{r-2} = 1$, then set $b \leftarrow 2^{r-1} - b$.
5. Output b.

A.2.7 Computing the order of a given integer modulo a prime

Let p be a prime and let g satisfy $1 < g < p$. The following algorithm determines the order of g modulo p when the factorization of $p - 1$ is known.

Input: a prime p and an integer g with $1 < g < p$.

Output: the order d of g modulo p.

1. Factor $p - 1 = \prod_i p_i^{e_i}$
2. For all divisors d of $p - 1$
   For all primes $p_i \mid d$
   If $g^d \equiv 1 \pmod{p}$ and $g^{d/p_i} \neq 1 \pmod{p}$
   Output d.

A.2.8 Constructing an integer of a given order modulo a prime

Let p be a prime and let T divide $p - 1$. The following algorithm generates an element of $GF(p)$ of order T when the factorization of $p - 1$ is known.

Input: a prime p and an integer T dividing $p - 1$.

Output: an integer u having order T modulo p.

1. Generate a random integer g between 1 and p.
2. Compute via A.2.7 the order d of g modulo p.
3. If T does not divide d, then go to step 1.
4. Output $u := g^{d/T} \bmod p$.

A.3 Extension fields: Overview

A.3.1 Finite fields

A finite field (or Galois field) is a set with finitely many elements in which the usual algebraic operations (addition, subtraction, multiplication, and division by nonzero elements) are possible and in which the usual algebraic laws (commutative, associative, and distributive) hold. The order of a finite field is the number of elements it contains. If $q > 1$ is an integer, then a finite field of order q exists if q is a prime power and not otherwise.

The finite field of a given order is unique in the sense that any two fields of order q display identical algebraic structure. Nevertheless, there are often many ways to represent a field. It is traditional to denote the finite field of order q by F_q or $GF(q)$; this standard uses the latter notation for typographical reasons. It should be borne in mind that the expressions "the field $GF(q)$" and "the field of order q" usually imply a choice of field representation.

In pairing-based cryptography, one makes use of $GF(p^n)$ for various n and p.

A.3.2 Polynomials over finite fields

A polynomial over $GF(q)$ is a polynomial with coefficients in $GF(q)$. Addition and multiplication of polynomials over $GF(q)$ are defined as usual in polynomial arithmetic, except that the operations on the coefficients are performed in $GF(q)$.

A polynomial over the prime field $GF(p)$ is commonly called a polynomial modulo p. Addition and multiplication are the same as for polynomials with integer coefficients, except that the coefficients of the results are reduced modulo p.

Example: Over the prime field $GF(7)$,

$$(t^2 + 4t + 5) + (t^3 + t + 3) = t^3 + t^2 + 5t + 1$$
$$(t^2 + 3t + 4)\,(t + 4) = t^3 + 2t + 2$$

A binary polynomial is a polynomial modulo 2.

Example: Over the field $GF(2)$,

$$(t^3 + 1) + (t^3 + t) = t + 1$$
$$(t^2 + t + 1)\,(t + 1) = t^3 + 1$$

A polynomial over $GF(q)$ is reducible if it is the product of two smaller degree polynomials over $GF(q)$; otherwise, it is irreducible. For instance, the examples in this subclause show that $t^3 + 2t + 2$ is reducible over $GF(7)$ and that the binary polynomial $t^3 + 1$ is reducible.

Every nonzero polynomial over $GF(q)$ has a unique representation as the product of powers of irreducible polynomials. (This result is analogous to the fact that every positive integer has a unique representation as the product of powers of prime numbers.) The degree – 1 factors correspond to the roots of the polynomial.

A.3.2.1 Polynomial congruences

Modular reduction and congruences can be defined among polynomials over $GF(q)$, in analogy to the definitions for integers given in A.1.1. To reduce a polynomial $a(t)$ modulo a nonconstant polynomial $m(t)$, one divides $a(t)$ by $m(t)$ by long division of polynomials and takes the remainder $r(t)$. This operation is written as

$$r(t) := a(t) \bmod m(t)$$

The remainder $r(t)$ shall either equal 0 or have degree smaller than that of $m(t)$.

If $m(t) = t - c$ for some element c of $GF(q)$, then $a(t) \bmod m(t)$ is just the constant $a(c)$.

Two polynomials $a(t)$ and $b(t)$ are said to be congruent modulo $m(t)$ if the two polynomials have the same result on reduction modulo $m(t)$. This relationship is written as

$$a(t) \equiv b(t) \pmod{m(t)}$$

One can define addition, multiplication, and exponentiation of polynomials (to integral powers) modulo $m(t)$, analogously to how the operations are defined for integer congruences in A.1.1. In the case of a prime field $GF(p)$, each of these operations involves both reduction of the polynomials modulo $m(t)$ and reduction of the coefficients modulo p.

A.3.3 Extension fields

If m is a positive integer, then the extension field $GF(p^m)$ consists of the p^m possible m-tuples of integers modulo p. Thus, for example,

$GF(2^3) = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$GF(3^2) = \{00, 01, 02, 10, 11, 12, 20, 21, 22\}$

The integer m is called the degree of the field.

A.3.3.1 Addition

For $m > 1$, the addition of two elements is implemented by component-wise addition modulo p. Thus, for example, in $GF(2^5)$, we have

$(11001) + (10100) = (01101)$

and in $GF(3^2)$, we have

$(01) + (22) = (20)$

A.3.3.2 Multiplication

There is more than one way to implement multiplication in $GF(p^m)$. To specify a multiplication rule, one chooses a basis representation for the field. The basis representation is a rule for interpreting each m-tuple; the multiplication rule follows from this interpretation.

For the purposes of this standard, we focus on polynomial basis representations.

A.3.4 Polynomial basis representations

In a polynomial basis representation, each element of $GF(p^m)$ is represented by a different polynomial modulo p of degree less than m. More explicitly, the tuple $(a_{m-1} \ldots a_2\, a_1\, a_0)$ is taken to represent the polynomial

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

where $0 \le a_i \le p-1$ for $0 \le i \le m-1$.

The polynomial basis is the set

$$B = \{t^{m-1}, \ldots, t^2, t, 1\}$$

The addition of m-tuples, as defined in A.3.3, corresponds to the addition of polynomials modulo p.

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

There is a polynomial basis representation for GF(p^m) corresponding to each irreducible polynomial $f(t)$ of degree m over GF(p). Irreducible polynomials modulo p exist of every degree. Roughly speaking, every one out of m polynomials modulo p of degree m is irreducible.

A.3.5 Extension fields (continued)

A.3.5.1 Exponentiation

If k is a positive integer and α is an element of $GF(p^m)$, then exponentiation is the operation of computing α^k. Subclause A.4.3 contains an efficient method for exponentiation.

A.3.5.2 Division

If α and $\beta \neq 0$ are elements of the field $GF(p^m)$, then the quotient α / β is the element γ such that $\alpha = \beta\gamma$.

In the finite field $GF(p^m)$, modular division is possible for any denominator other than 0. The set of nonzero elements of $GF(p^m)$ is denoted $GF(p^m)^*$.

Subclause A.4.2 contains an efficient method for division.

A.3.5.3 Orders

The order of an element γ of $GF(p^m)^*$ is the smallest positive integer v such that $\gamma^v = 1$. The order always exists and divides $p^m - 1$. If k and l are integers, then $\gamma^k = \gamma^l$ in $GF(p^m)$ if and only if $k \equiv l \pmod{v}$.

A.3.5.4 Generators

If v divides $p^m - 1$, then there exists an element of $GF(p^m)^*$ having order v. In particular, there always exists an element γ of order $p^m - 1$ in $GF(p^m)^*$. Such an element is called a generator for $GF(p^m)^*$ because every element of $GF(p^m)^*$ is some power of γ.

A.3.5.5 Exponentiation and discrete logarithms

Suppose that the element γ of $GF(p^m)^*$ has order v. Then an element η of $GF(p^m)^*$ satisfies $\eta = \gamma^l$ for some l if and only if $\eta^v = 1$. The exponent l is called the discrete logarithm of η (with respect to the base γ). The discrete logarithm is an integer modulo v.

A.3.5.6 Field extensions

Given two extensions $K = GF(p^n)$ and $L = GF(p^m)$, L is an extension of K if and only if $n \mid m$. For pairing-based cryptography, we often require that K be embedded in the extension L. This is defined in A.5.7.

A.4 Extension fields: Algorithms

The following algorithms perform operations in a finite field $GF(p^m)$ having p^m elements. The elements of $GF(p^m)$ are represented by a polynomial basis modulo the irreducible polynomial $f(t)$.

A.4.1 Exponentiation

Exponentiation can be performed efficiently by the binary method outlined as follows.

Input: a positive integer k, a field $GF(p^m)$ and a field element α.

Output: α^k.

1. Let $k = k_r\, k_{r-1} \ldots k_1\, k_0$ be the binary representation of k, where the most significant bit k_r of k is 1.
2. Set $x \leftarrow \alpha$.
3. For i from $r - 1$ down to 0, do the following:
   3.1 Set $x \leftarrow x^2$.
   3.2 If $k_i = 1$, then set $x \leftarrow \alpha\, x$.
4. Output x.

There are several modifications that improve the performance of this algorithm. These methods are summarized in Gordon [B68].

A.4.2 Division

The quotient α/β can be computed directly (i.e., in one step by an algorithm with inputs α and β), or indirectly (by computing the multiplicative inverse β^{-1} and then multiplying it by α). The common method of performing division in a finite field $GF(p^m)$ is the indirect method using *The Extended Euclidean Algorithm*.

Input: two polynomials $f(x)$, $g(x) \neq 0$ over $GF(p^m)$.

Output: $d(x) = \mathrm{GCD}\,(f(x), g(x))$, $s(x)$, $t(x)$ satisfying $s(x)f(x)+t(x)g(x) = d(x)$.

1. Set $s_1(x) \leftarrow 1$, $s_2(x) \leftarrow 0$, $t_1(x) \leftarrow 1$, $t_2(x) \leftarrow 0$.
2. While $g(x) \neq 0$
   2.1 Set $q(x) \leftarrow \lfloor f(t) / g(t) \rfloor$, $r(x) \leftarrow f(x) - g(x)q(x)$.
   2.2 Set $s(x) \leftarrow s_2(x) - q(x)s_1(x)$, $t(x) \leftarrow t_2(x) - q(x)t_1(x)$.
   2.3 Set $f(x) \leftarrow g(x)$, $g(x) \leftarrow r(x)$.
   2.4 Set $s_2(x) \leftarrow s_1(x)$, $s_1(x) \leftarrow s(x)$, $t_2(x) \leftarrow t_1(x)$, $t_1(x) \leftarrow t(x)$.
3. Set $d(x) \leftarrow f(x)$, $s(x) \leftarrow s_2(x)$, $t(x) \leftarrow t_2(x)$.
4. Output $d(x)$, $s(x)$, $t(x)$.

This algorithm produces the $t(x)$, the multiplicative inversion of $g(x)$ modulo $f(x)$. By $\lfloor f(x) / g(x) \rfloor$ is meant the quotient upon polynomial division, dropping any remainder.

A.4.3 Squares

To determine whether a given element is a square, the Legendre symbol can be computed as follows:

Input: $f(x)$, $g(x) \neq 0 \in GF(p^m)$ where $g(x)$ is irreducible.

Output: the Legendre–Kronecker–Jacobi symbol $(f(x) / g(x))$.

1. Set $k \leftarrow 1$.
2. While $\deg(m) \neq 0$
   - 2.1. If $f(x) = 0$, then return 0.
   - 2.2. $a \leftarrow$ the leading coefficient of $f(x)$.
   - 2.3. $f(x) \leftarrow f(x) / a$.
   - 2.4. If $\deg(m) \equiv 1 \pmod 2$, then $k \leftarrow k\ (a / p)$.
   - 2.5. If $p^{\deg(m)} \equiv 3 \pmod 4$ and $\deg(m) \deg(f) \equiv 1 \pmod 2$, then $k \leftarrow -k$.
   - 2.6. $r(x) \leftarrow f(x)$, $f(x) \leftarrow m(x) \bmod r(x)$, $m(x) \leftarrow r(x)$.
3. Return k.

A.4.4 Square roots

To compute a square root in a finite field, the Tonelli-Shanks algorithm is used.

Input: an element $a \in GF(q)$, where $q = p^k$.

Output: an element $x \in GF(q)$ such that $a = x^2$ or "quadratic nonresidue" if a is not a square.

1. Write $q - 1 = 2^e v$.
2. Choose $n \in GF(q)$ until $(n / q) = -1$. Set $z \leftarrow n^v$.
3. Set $y \leftarrow z$, $r \leftarrow e$, $x \leftarrow a^{((v-1)/2)}$, $b \leftarrow ax^2$, $x \leftarrow ax$.
4. If $b = 1$, return x.

   Otherwise, find the smallest integer m such that $b^{2^m} = 1$. If $m = r$, return "a is a quadratic nonresidue."
5. Set $t \leftarrow y^{2^{r-m-1}}$, $y \leftarrow t^2$, $r \leftarrow m$, $x \leftarrow xt$, $b \leftarrow by$ and go to step 4.

Notes: p, k, q, e, v, r, m are integers.

a, x, n, z, y, b, t are elements $\in GF(q)$.

(n / q) can be computed by forming $n^{(q-1)/2}$.

A.4.5 Trace in binary field extension

If α is an element of $GF(2^m)$, the trace of α is

$$\mathrm{Tr}(\alpha) = \alpha + \alpha^2 + \alpha^{2^2} + \ldots + \alpha^{2^{m-1}}$$

The value of $\mathrm{Tr}(\alpha)$ is 0 for half the elements of $GF(2^m)$, and 1 for the other half.

The trace can be computed efficiently as follows.

The basic algorithm inputs $\alpha \in GF(2^m)$ and outputs $T = \mathrm{Tr}(\alpha)$.

1. Set $T \leftarrow \alpha$.
2. For i from 1 to $m - 1$, do the following:
   2.1 $T \leftarrow T^2 + \alpha$.
3. Output T.

If many traces are to be computed with respect to a fixed polynomial basis

$$\{t^{m-1}, \ldots, t, 1\}$$

then it is more efficient to compute and store the element

$$\tau = (\tau_{m-1} \ldots \tau_1 \tau_0)$$

where each coordinate

$$\tau_j = \mathrm{Tr}(t^j)$$

is computed via the basic algorithm. Subsequent traces can be computed via

$$\mathrm{Tr}(\alpha) = \alpha \cdot \tau$$

where the "dot product" of the bit strings is given by bitwise AND (or bitwise multiplication).

A.4.6 Half-trace in binary fields

If m is odd, then the half-trace of $\alpha \in GF(2^m)$ is

$$\mathrm{HfTr}(\alpha) = \alpha + \alpha^{2^2} + \alpha^{2^4} + \ldots + \alpha^{2^{m-1}}$$

The following algorithm inputs $\alpha \in GF(2^m)$ and outputs $H = \mathrm{HfTr}(\alpha)$:

1. Set $H \leftarrow \alpha$.
2. For i from 1 to $(m - 1)/2$, do the following:
   2.1 $H \leftarrow H^2$.
   2.2 $H \leftarrow H^2 + \alpha$.
3. Output H.

A.4.7 Solving quadratic equations over $GF(2^m)$

If β is an element of $GF(2^m)$, then the equation

$$z^2 + z = \beta$$

has $2 - 2T$ solutions over $GF(2^m)$, where $T = \mathrm{Tr}(\beta)$. Thus, there are either 0 or 2 solutions. If z is one solution, then the other solution is $z + 1$. In the case $\beta = 0$, the solutions are 0 and 1.

The following algorithms compute a solution if one exists:

Input: a field $GF(2^m)$ along with a polynomial or normal basis for representing its elements; an element $\beta \neq 0$.

Output: an element z for which $z^2 + z = \beta$, if such an element exists.

If m is odd, then compute z := half-trace of β via A.4.6. For m even, proceed as follows:

1. Choose random $\rho \in GF(2^m)$.
2. Set $z \leftarrow 0$ and $w \leftarrow \rho$.
3. For i from 1 to $m - 1$, do the following:
   3.1 Set $z \leftarrow z^2 + w^2 \beta$.
   3.2 Set $w \leftarrow w^2 + \rho$.
4. If $w = 0$, then go to step 1.
5. Output z.

If the latter algorithm is to be used repeatedly for the same field and memory is available, then it is more efficient to precompute and store ρ and the values of w. Any element of trace 1 will serve as ρ, and the values of w depend only on ρ and not on β.

The algorithm presented here produces a solution z provided that one exists. If it is unknown whether a solution exists, then the output z should be checked by comparing $\gamma := z^2 + z$ with β. If $\gamma = \beta$, then z is a solution; otherwise, no solutions exist.

A.4.8 Trace in ternary field extensions

The following algorithm computes the trace of $\alpha \in GF(3^m)$:

Input: $\alpha \in GF(3^m)$

Output: $T = \mathrm{Tr}(\alpha)$

1. Set $T \leftarrow \alpha$.
2. For i from 1 to $m - 1$, do the following:
   2.1 $T \leftarrow T^3 + \alpha$.
3. Output T.

A.4.9 The 1/3-trace in ternary fields

If $r = m \bmod 3$ is not zero, then the 1/3-trace of $\alpha \in GF(3^m)$ is $CrTr(\alpha) = \alpha + \alpha^{3^3} + \alpha^{3^6} + ... + \alpha^{3^{m-r}}$. The following algorithm calculates $CrTr(\alpha)$.

Input: $\alpha \in GF(3^m)$.

Output: $C = CrTr(\alpha)$.

1. Set $C \leftarrow \alpha$.
2. For i from 1 to $(m - r)/3$, do the following:

2.1 $C \leftarrow C^{3^3} + \alpha$.

3. Output *C*.

A.4.10 Solving cubic equations over *GF*(3^m)

The following algorithm computes a solution to $z^3 - z = \beta$ over $GF(3^m)$, where *m* is not divisible by 3.

Input: $\beta \in GF(3^m)$ with $Tr(\beta) \neq 0$.

Output: $z \in GF(3^m)$ with $z^3 - z = \beta$.

1. Set $C \leftarrow CrTr(\beta)$.
2. Set $z \leftarrow C^3 - C$.
3. If $m \equiv 1 (\mathrm{mod}\, 3)$, then $z \leftarrow \beta - z$.
4. Output *z*.

A.5 Polynomials over a finite field

The computations in this clause can take place either over a prime field (having a prime number *p* of elements) or over a binary field (having 2^m elements) or over a ternary field (having 3^m elements).

A.5.1 Exponentiation modulo a polynomial

If *k* is a positive integer and *f*(*t*) and *m*(*t*) are polynomials with coefficients in the field *GF*(*q*), then $f(t)^k$ mod *m*(*t*) can be computed efficiently by the *binary method* outlined as follows.

Input: a positive integer *k*, a field *GF*(*q*), and polynomials *f*(*t*) and *m*(*t*) with coefficients in *GF*(*q*).

Output: the polynomial $f(t)^k$ mod *m*(*t*).

1. Let $k = k_r\, k_{r-1} \ldots k_1\, k_0$ be the binary representation of *k*, where the most significant bit k_r of *k* is 1.
2. Set $u(t) \leftarrow f(t) \bmod m(t)$.
3. For *i* from *r* – 1 downto 0, do the following:
   3.1 Set $u(t) \leftarrow u(t)^2 \bmod m(t)$.
   3.2 If $k_i = 1$, then set $u(t) \leftarrow u(t)\, f(t) \bmod m(t)$.
4. Output *u*(*t*).

There are several modifications that improve the performance of this algorithm. These methods are summarized in Gordon [B68].

A.5.2 GCDs over a finite field

If *f*(*t*) and $g(t) \neq 0$ are two polynomials with coefficients in the field *GF*(*q*), then there is a unique monic polynomial *d*(*t*) of largest degree that divides both *f*(*t*) and *g*(*t*). The polynomial *d*(*t*) is called the GCD of *f*(*t*) and *g*(*t*). The following algorithm computes the GCD of two polynomials:

Input: a finite field *GF*(*q*) and two polynomials *f*(*t*), $g(t) \neq 0$ over *GF*(*q*).

Output: *d*(*t*) = GCD(*f*(*t*), *g*(*t*)).

1. Set $a(t) \leftarrow f(t)$, $b(t) \leftarrow g(t)$.
2. While $b(t) \neq 0$
   2.1 Set $c(t) \leftarrow$ the remainder when $a(t)$ is divided by $b(t)$.
   2.2 Set $a(t) \leftarrow b(t)$.
   2.3 Set $b(t) \leftarrow c(t)$.
3. Set $\alpha \leftarrow$ the leading coefficient of $a(t)$.
4. Set $d(t) \leftarrow \alpha^{-1} a(t)$.
5. Output $d(t)$.

A.5.3 Factoring polynomials over *GF*(*p*) (special case)

Let $f(t)$ be a polynomial with coefficients in the field $GF(p)$, and suppose that $f(t)$ factors into distinct irreducible polynomials of degree d. (This is the special case needed in A.9.) The following algorithm finds a random degree-d factor of $f(t)$ efficiently:

Input: a prime $p > 2$, a positive integer d, and a polynomial $f(t)$, which factors modulo p into distinct irreducible polynomials of degree d.

Output: a random degree-d factor of $f(t)$.

1. Set $g(t) \leftarrow f(t)$.
2. While $\deg(g) > d$
   2.1 Choose $u(t) \leftarrow$ a random monic polynomial of degree $2d - 1$.
   2.2 Compute via A.5.1
   $$c(t) := u(t)^{(p^d-1)/2} \bmod g(t).$$
   2.3 Set $h(t) \leftarrow \mathrm{GCD}(c(t) - 1, g(t))$.
   2.4 If $h(t)$ is constant or $\deg(g) = \deg(h)$, then go to step 2.1.
   2.5 If $2\deg(h) > \deg(g)$, then set $g(t) \leftarrow g(t) / h(t)$; else $g(t) \leftarrow h(t)$.
3. Output $g(t)$.

A.5.4 Factoring polynomials over *GF*(2) (special case)

Let $f(t)$ be a polynomial with coefficients in the field $GF(2)$ and suppose that $f(t)$ factors into distinct irreducible polynomials of degree d. (This is the special case needed in A.9.) The following algorithm finds a random degree-d factor of $f(t)$ efficiently:

Input: a positive integer d and a polynomial $f(t)$ that factors modulo 2 into distinct irreducible polynomials of degree d.

Output: a random degree-d factor of $f(t)$.

1. Set $g(t) \leftarrow f(t)$.
2. While $\deg(g) > d$
   2.1 Choose $u(t) \leftarrow$ a random monic polynomial of degree $2d - 1$.
   2.2 Set $c(t) \leftarrow u(t)$.
   2.3 For i from 1 to $d - 1$, do the following:
      2.3.1 $c(t) \leftarrow c(t)^2 + u(t) \bmod g(t)$.
   2.4 Compute $h(t) := \mathrm{GCD}(c(t), g(t))$ via A.5.2.
   2.5 If $h(t)$ is constant or $\deg(g) = \deg(h)$, then go to step 2.1.
   2.6 If $2\deg(h) > \deg(g)$, then set $g(t) \leftarrow g(t) / h(t)$; else $g(t) \leftarrow h(t)$.
3. Output $g(t)$.

A.5.5 Checking polynomials over $GF(2^r)$ for irreducibility

If $f(t)$ is a polynomial with coefficients in the field $GF(2^r)$, then $f(t)$ can be tested efficiently for irreducibility using the following algorithm:

Input: a polynomial $f(t)$ with coefficients in $GF(2^r)$.

Output: the message "true" if $f(t)$ is irreducible; the message "false" otherwise.

1. Set $d \leftarrow$ degree of $f(t)$.
2. Set $u(t) \leftarrow t$.
3. For i from 1 to $\lfloor d/2 \rfloor$, do the following:
   - 3.1 For j from 1 to r, do the following:
     Set $u(t) \leftarrow u(t)^2 \bmod f(t)$.
     Next j.
   - 3.2 Set $g(t) \leftarrow \mathrm{GCD}(u(t) + t, f(t))$.
   - 3.3 If $g(t) \neq 1$, then output "false" and stop.
3. Output "true."

A.5.6 Finding a root in $GF(2^m)$ of an irreducible binary polynomial

If $f(t)$ is an irreducible polynomial modulo 2 of degree d dividing m, then $f(t)$ has d distinct roots in the field $GF(2^m)$. A random root can be found efficiently using the following algorithm:

Input: an irreducible polynomial modulo 2 of degree d, and a field $GF(2^m)$, where d divides m.

Output: a random root of $f(t)$ in $GF(2^m)$.

1. Set $g(t) \leftarrow f(t)$ ($g(t)$ is a polynomial over $GF(2^m)$).
2. While $\deg(g) > 1$
   - 2.1 Choose random $u \in GF(2^m)$.
   - 2.2 Set $c(t) \leftarrow ut$.
   - 2.3 For i from 1 to $m - 1$, do the following:
     2.3.1 $c(t) \leftarrow c(t)^2 + ut \bmod g(t)$.
   - 2.4 Set $h(t) \leftarrow \mathrm{GCD}(c(t), g(t))$.
   - 2.5 If $h(t)$ is constant or $\deg(g) = \deg(h)$, then go to step 2.1.
   - 2.6 If $2\deg(h) > \deg(g)$, then set $g(t) \leftarrow g(t) / h(t)$; else $g(t) \leftarrow h(t)$.
3. Output $g(0)$.

A.5.7 Embedding in an extension field

Given a field $\mathbf{F} = GF(p^d)$, the following algorithm embeds $\mathbf{F}$ into an extension field $\mathbf{K} = GF(p^{de})$:

Input: integers d and e; a polynomial basis B for $\mathbf{F} = GF(p^d)$ with field polynomial $f(t)$; a polynomial basis for $\mathbf{K} = GF(p^{de})$.

Output: an embedding of $\mathbf{F}$ into $\mathbf{K}$; that is a function taking each $\alpha \in \mathbf{F}$ to a corresponding element β of $\mathbf{K}$.

1. Compute via A.5.6 a root $\lambda \in \mathbf{K}$ of $f(t)$.
2. Output

$$\beta := a_{m-1}\lambda^{m-1} + \ldots + a_2\lambda^2 + a_1\lambda + a_0$$

where $(a_{m-1} \ldots a_1\ a_0)$ is the m-tuple representing α with respect to β.

A.6 Elliptic curves: Overview

A.6.1 Introduction

A plane curve is defined to be the set of points satisfying an equation $F(x, y) = 0$. The simplest plane curves are lines (whose defining equation has degree 1 in x and y) and conic sections (degree 2 in x and y). The next simplest are the cubic curves (degree 3). These include elliptic curves, so called because of their origin problem of computing an arc length of an ellipse. This standard restricts its attention to cubic plane curves, although other representations could be defined. The coefficients of such a curve shall satisfy a side condition to guarantee the mathematical property of nonsingularity. The side condition is given in this clause for each family of curves.

An elliptic curve is a nonsingular (smooth) algebraic curve of genus one with a defined point. The set of points on an elliptic curve is topologically equivalent to a torus—a surface with one hole in it—and simplistically, the number of holes in a surface is the definition of the term "genus." Elliptic curves should strictly be written as a pair (E, $\mathcal{O}$), where E is the curve and $\mathcal{O}$ is the defined point. However, $\mathcal{O}$ is invariably taken to be the point at infinity and the elliptic curve is often simply referred to as E. (see Silverman [B149] for a mathematically precise definition of "elliptic curve.")

In cryptography, the elliptic curves of interest are those defined over finite fields. That is, the coefficients of the defining equation $F(x, y) = 0$ are elements of $GF(q)$, and the points on the curve are of the form $P = (x, y)$, where x and y are elements of $GF(q)$. Examples are given in A.6.1.1 through A.6.1.4.

A.6.1.1 The Weierstrass equation

There are several kinds of defining equations for elliptic curves, but the most common are the *Weierstrass equations*. This standard will be concerned with both ordinary and supersingular curves. The general equation of an elliptic curve is

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

If we let p denote the characteristic of K, the equation can be simplified for different values of p.

— For the finite fields $GF(p^m)$ with $p > 3$, the standard Weierstrass equation for ordinary curves is

$$y^2 = x^3 + ax + b$$

where a and b are integers modulo p for which $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

— For the binary finite fields $GF(2^m)$, the standard Weierstrass equation for ordinary curves is

$$y^2 + xy = x^3 + ax^2 + b$$

where a and b are elements of $GF(2^m)$ with $b \neq 0$.

— One can also define the following supersingular curves. We define these by giving the base field, then the "embedding degree" k (which will be used later), followed by the equation of the curve. We will only be interested in even embedding degree curves.

1) $GF(2^s)$, $k = 2$, s even, $y^2 + y = x^3 + \delta x$, where Tr $\delta \neq 0$.

2) $GF(2^s)$, $k = 2$, s odd, $y^2 + y = x^3$.

3) $GF(3^s)$ with $k = 2$, $y^2 = x^3 + ax + b$ if $j = 0$, $y^2 = x^3 + ax^2 + b$ if $j \neq 0$.

4) $GF(p)$ with $p > 3$, $k = 2$, $y^2 = x^3 + ax + b$

   i) If $p \equiv 3 \pmod 4$, then $b = 0$ and $-a$ is not a square mod p.

   ii) If $p \equiv 5 \pmod 6$, then $a = 0$.

   iii) If $p \equiv 1 \pmod{12}$, then $a = 3mc^2$, $b = 2mc^3$, where $m = j / (1728 - j)$ and $c \in GF(p)^\times$.

5) $GF(2^s)$, $k = 4$, $y^2 + y = x^3 + x + b$ for $b = 0$ or 1.

6) $GF(3^s)$, $k = 6$, $y^2 = x^3 - x + b$ for $b = \pm 1$.

Given a Weierstrass equation, the elliptic curve E consists of the solutions (x, y) over $GF(q)$ to the defining equation, along with an additional element called the point at infinity (denoted $\mathcal{O}$). The points other than $\mathcal{O}$ are called finite points. The number of points on E (including $\mathcal{O}$) is called the order of E and is denoted by $\#E\,(GF(q))$.

Example: Let E be the curve

$$y^2 = x^3 + 10x + 5$$

over the field $GF(13)$. Then the points on E are

$$\{\mathcal{O}, (1,4), (1,9), (3,6), (3,7), (8,5), (8,8), (10,0), (11,4), (11,9)\}$$

Thus, the order of E is $\#E\,(GF(13)) = 10$.

Example: Let E be the curve

$$y^2 + xy = x^3 + (t + 1)x^2 + 1$$

over the field $GF(2^3)$ given by the polynomial basis with field polynomial $t^3 + t + 1 = 0$. Then the points on E are

$\{\mathcal{O}, ((000), (001)),$
$((010), (100)), ((010), (110)), ((011), (100)), ((011), (111)),$
$((100), (001)), ((100), (101)), ((101), (010)), ((101), (111)),$
$((110), (000)), ((110), (110)), ((111), (001)), ((111), (110))\}$

Thus, the order of E is $\#E\,(GF(2^3)) = 14$.

For more information on elliptic curve cryptography, see Menezes [B114].

A.6.1.2 Orders

The order of a point P on an elliptic curve is the smallest positive integer r such that $rP = \mathcal{O}$. The order always exists and divides the order of the curve $\#E(GF(q))$. If k and l are integers, then $kP = lP$ if and only if $k \equiv l \pmod r$.

A.6.1.3 Pairings

The primitives defined in the body of this standard use the general concept of a pairing. Here, we define a pairing e as a bilinear map between elements of two finite, cyclic, additive groups, G_1 and G_2 to a third finite, cyclic group G_T defined multiplicatively:

$$e: G_1 \times G_2 \rightarrow G_T$$

Both G_1 and G_2 are of prime order r, as is G_T. The bilinear property is such that

For all P, P' $\in$ G_1 and all Q, Q' $\in$ G_2, e(P + P',Q) = e(P,Q)e(P',Q) and e(P,Q + Q') = e(P,Q) e(P,Q')

We also impose the condition that the map be nondegenerate; i.e.:

For all $0 \neq P \in G_1$ there exists $Q \in G_2$ such that $e(P,Q) \neq 1$

For all $0 \neq Q \in G_2$ there exists $P \in G_1$ such that $e(P,Q) \neq 1$

For cryptographic use, the groups G_1 and G_2 over which the pairings are defined are subgroups of points on an elliptic curve.

Elliptic curves fall into two general categories: supersingular curves and ordinary curves. The former are curves where the kernel of the "multiplication by p" map (where p is the characteristic of K) is trivial. Supersingular curves were the first to be considered for use in pairing-based cryptography because all supersingular curves possess maps (non-$GF(q)$-rational endomorphisms) that prove useful in constructing the Tate pairing.

The first pairing-based cryptosystems used the Weil and Tate pairings on supersingular curves. Further research in pairing-based cryptography has led to a range of suitable pairings and families of curves. Apart from super-singular curves, all the curve families are of ordinary curves. The primary factors that dictate which pairing and curve to use are the efficiency of computations and the security level. As with most cryptography, increasing levels of security can be obtained by increasing the size of the field over which the operations are defined, but not all curve-pairing combinations have the same relationship between security and efficiency. Subclause A.12 suggests some appropriate system parameters for different levels of security and suggests appropriate combinations of pairings and curves.

A.6.1.4 Twists

For a field K, if char $(K) \neq 2,3$, then we define quadratic, quartic, and sextic twists of $E(K)$ as follows. Let

$$E: y^2 = x^3 + Ax + B$$

Case 1: if $A, B \neq 0$, then there are quadratic twists only; one can define the twist by giving a value D to produce the curve

$$E': y^2 = x^3 + D^2Ax + D^3B$$

Essentially, there are two such values of D, one producing the original curve and one producing the quadratic twist.

Case 2: if $B = 0$, then there are quartic twists and quadratic twists. By giving a value D, one can define the quartic twists by

$$E'\colon y^2 = x^3 + DAx$$

There are essentially four such values of D, which produce nonisomorphic curves over the base field. One of these produces the same curve, two the quartic twists, and the remaining one produces the quadratic twist of E.

Case 3: if $A = 0$, there are sextic twists and quadratic twists. By giving a suitable value of D, one can define the twists via

$$E'\colon y^2 = x^3 + DB$$

There are essentially six such values of D: one produces the curve itself, one produces the quadratic twist, two produce a cubic twist, and the remaining two produce sextic twists.

A.6.2 Operations on elliptic curves

There is an addition operation on the points of an elliptic curve that possesses the algebraic properties of ordinary addition (e.g., commutativity and associativity). This operation can be described geometrically as follows.

Define the *inverse* of the point $P = (x, y)$ to be

$$-P = \begin{cases} (x, -y) & \text{if } p \geq 3 \\ (x, x + y) & \text{if } q = 2^m \text{ and } E \text{ is ordinary} \\ (x, c + y) & \text{if } q = 2^m \text{ and } E \text{ is supersingular} \end{cases}$$

Then the sum $P + Q$ of the points P and Q is the point R with the property that P, Q, and $-R$ lie on a common line.

A.6.2.1 The point at infinity

The point at infinity $\mathcal{O}$ plays a role analogous to that of the number 0 in ordinary addition. Thus

$$P + \mathcal{O} = P$$
$$P + (-P) = \mathcal{O}$$

for all points P.

A.6.2.2 Full addition

When implementing the formulas for elliptic curve addition, it is necessary to distinguish between doubling (adding a point to itself) and adding two distinct points that are not inverses of each other because the formulas are different in the two cases. Also, there are the special cases involving $\mathcal{O}$. By "full addition" is meant choosing and implementing the appropriate formula for the given pair of points. Algorithms for full addition are given in A.7.1, A.7.2, A.7.3, and A.7.9.

A.6.2.3 Scalar multiplication

Elliptic curve points can be added but not multiplied. It is, however, possible to perform scalar multiplication, which is another name for repeated addition of the same point. If n is a positive integer and P a point on an elliptic curve, then the scalar multiple nP is the result of adding n copies of P. Thus, for example, $5P = P + P + P + P + P$.

The notion of scalar multiplication can be extended to zero and the negative integers via

$$0P = \mathcal{O},\ (-n)\,P = n\,(-P)$$

A.6.3 Curve orders

Finding a base point of prime order requires knowledge of the curve order $n = \#E(GF(q))$. Since r shall divide n, one has the following problem: *given a field* $\mathbf{F} = GF(q)$*, find an elliptic curve defined over* $\mathbf{F}$ *whose order is divisible by a sufficiently large prime r.* (Note that "sufficiently large" is defined in terms of the desired security; see A.12.) This subclause discusses this problem.

A.6.3.1 Basic facts

— If n is the order of an elliptic curve over $GF(q)$, then the Hasse bound is

$$q - 2\sqrt{q} + 1 \le q \le q + 2\sqrt{q} + 1$$

Thus, the order of an elliptic curve over $GF(q)$ is approximately q.

— If q is a prime p, let n be the order of the curve $y^2 = x^3 + ax + b$, where a and b are both nonzero. Then if $\lambda \neq 0$, the order of the curve $y^2 = x^3 + a\lambda^2 x + b\lambda^3$ is n if λ is a square modulo p and $2p + 2 - n$ otherwise. (This fact allows one to replace a given curve by one with the same order and satisfying some extra condition, such as $a = p - 3$, which will be used in A.7.5.) In the case $b = 0$, there are four possible orders; in the case $a = 0$, there are six. (The formulas for these orders can be found in step 6 of A.9.2.3.)

— If $q = 2^m$, let n be the order of the curve $y^2 + xy = x^3 + ax^2 + b$, where a and b are both nonzero. Then, if $\lambda \neq 0$, the order of the curve $y^2 + xy = x^3 + (a + \lambda)\,x^2 + b$ is n if λ has trace 0 and $2^{m+1} + 2 - n$ otherwise (see A.4.5). (This fact allows one to replace a given curve by one with the same order and satisfying some extra condition, such as $a = 0$ which will be used in A.7.8.)

— If $q = 2^m$, then the curves $y^2 + xy = x^3 + ax^2 + b$ and $y^2 + xy = x^3 + a^2 x^2 + b^2$ have the same order.

A.6.3.2 Near primality

Given a trial division bound l_{max}, the positive integer k is called smooth if every prime divisor of k is at most l_{max}. Given large positive integers r_{min} and r_{max}, u is called nearly prime if $u = kr$ for some prime r in the interval $r_{min} \le r \le r_{max}$ and some smooth integer k. (The requirement that k be smooth is omitted in most definitions of near primality. It is included here to guarantee that there exists an efficient algorithm to check for near primality.) In the case in which a prime order curve is desired, the bound l_{max} is set to 1.

NOTE—Because all elliptic curves over $GF(q)$ have order at most $u_{max} = q + 2\sqrt{q}$, then r_{max} should be no greater than u_{max}. (If no maximum is desired, e.g., as in ANSI X9.62-2005 [B9], then one takes $r_{max} \leftarrow u_{max}$.) Moreover, if r_{min} is close to u_{max}, then there will be a small number of possible curves to choose from, so that finding a suitable one will be more difficult. If a prime-order curve is desired, then a convenient choice is $r_{min} = q + \sqrt{q}$.

A.6.4 Representation of points

This subclause discusses the issues involved in choosing representations for points on elliptic curves for purposes of internal computation and for external communication.

A.6.4.1 Affine coordinates

A finite point on E is specified by two elements x, y in $GF(q)$ satisfying the defining equation for E. These are called the affine coordinates for the point. The point at infinity $\mathcal{O}$ has no affine coordinates. For the purposes of internal computation, it is most convenient to represent $\mathcal{O}$ by a pair of coordinates (x, y) not on E. For $q = 2^m$, the simplest choice is $\mathcal{O} = (0,0)$. For $q = p^m$, one chooses $\mathcal{O} = (0,0)$ unless $b = 0$; in which case, $\mathcal{O} = (0,1)$.

A.6.4.2 Coordinate compression

The affine coordinates of a point require $2ml$ bits to store and transmit where q itself requires l bits to represent it. This is far more than is needed, however. For purposes of external communication, therefore, it can be advantageous to compress one or both of the coordinates.

The y coordinate can always be compressed. The compressed y coordinate, denoted $\tilde{y}$, is a single bit, defined as follows:

— If q is a power of an odd prime, then $\tilde{y} := y \bmod 2$, where y is interpreted as a positive integer less than q. Put another way, $\tilde{y}$ is the rightmost bit of y.

— If q is a power of 2, then $\tilde{y}$ is the rightmost bit of the field element $y\,x^{-1}$ (except when $x = 0$; in which case, $\tilde{y} := 0$).

NOTE 1—Algorithms for decompressing coordinates are given in A.7.11 and A.7.12.

NOTE 2—There are many other possible ways to compress coordinates; the methods given here are the ones that have appeared in the literature (see Menezes [B113] and Seroussi [B146]).

A.6.4.3 Projective coordinates

If division within $GF(q)$ is relatively expensive, then it may pay to keep track of numerators and denominators separately. In this way, one can replace division by α with multiplication of the denominator by α. This is accomplished by the projective coordinates X, Y, and Z, given by

$$x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}$$

The projective coordinates of a point are not unique because

$$(X, Y, Z) = (\lambda^2 X, \lambda^3 Y, \lambda Z)$$

for every nonzero $\lambda \in GF(q)$.

The projective coordinates of the point at infinity are $(\lambda^2, \lambda^3, 0)$, where $\lambda \neq 0$.

Other kinds of projective coordinates exist, but the ones given here provide the fastest arithmetic on elliptic curves. (See Chudnovsky and Chudnovsky [B44].)

These formulas provide the method for converting a finite point from projective coordinates to affine. To convert from affine to projective, one proceeds as follows:

$$X \leftarrow x, Y \leftarrow y, Z \leftarrow 1$$

Projective coordinates are well suited for internal computation but not for external communication because their use requires so many bits. The use of projective coordinates is more common over $GF(p)$ because division tends to be more expensive there.

A.7 Elliptic curves: General algorithms

A.7.1 Full addition and subtraction (prime case)

The following algorithm implements a full addition (on a curve modulo p) in terms of affine coordinates. Note that this algorithm can also be used for supersingular curves of characteristic 3.

Input: a field $K = GF(p^n)$ for $p > 3$; coefficients a, b for an elliptic curve E: $y^2 = x^3 + ax + b$ over K; 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 $P_0 = \mathcal{O}$, then output $P_2 \leftarrow P_1$ and stop.
2. If $P_1 = \mathcal{O}$, then output $P_2 \leftarrow P_0$ and stop.
3. If $x_0 \neq x_1$, then
   3.1 Set $\lambda \leftarrow (y_0 - y_1) / (x_0 - x_1) \bmod p$.
   3.2 Go to step 7.
4. If $y_0 \neq y_1$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
5. If $y_1 = 0$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
6. Set $\lambda \leftarrow (3 x_1^2 + a) / (2y_1) \bmod p$.
7. Set $x_2 \leftarrow \lambda^2 - x_0 - x_1 \bmod p$.
8. Set $y_2 \leftarrow (x_1 - x_2) \lambda - y_1 \bmod p$.
9. Output $P_2 \leftarrow (x_2, y_2)$.

The preceding algorithm requires three or four modular multiplications and a modular inversion.

To subtract the point $P = (x, y)$, one adds the point $-P = (x, -y)$.

A.7.2 Full addition and subtraction (binary case)

The following algorithm implements a full addition [on an ordinary curve over $GF(2^m)$] in terms of affine coordinates.

Input: a field $GF(2^m)$; coefficients a, b for an elliptic curve E: $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^m)$; 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 $P_0 = \mathcal{O}$, then output $P_2 \leftarrow P_1$ and stop.
2. If $P_1 = \mathcal{O}$, then output $P_2 \leftarrow P_0$ and stop.
3. If $x_0 \neq x_1$ then
   3.1 Set $\lambda \leftarrow (y_0 + y_1) / (x_0 + x_1)$.
   3.2 Set $x_2 \leftarrow a + \lambda^2 + \lambda + x_0 + x_1$.
   3.3 Go to step 7.
4. If $y_0 \neq y_1$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
5. If $x_1 = 0$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
6. Set
   6.1 $\lambda \leftarrow x_1 + y_1 / x_1$.
   6.2 $x_2 \leftarrow a + \lambda^2 + \lambda$.
7. $y_2 \leftarrow (x_1 + x_2)\lambda + x_2 + y_1$.
8. $P_2 \leftarrow (x_2, y_2)$.

The preceding algorithm requires two general multiplications, a squaring, and a multiplicative inversion.

To subtract the point $P = (x, y)$, one adds the point $-P = (x, x + y)$.

A.7.3 Full addition and subtraction (supersingular curves in characteristic 2)

The following algorithm implements a full addition (on a supersingular curve over $GF(2^m)$) in terms of affine coordinates.

Input: a field $GF(2^m)$; coefficients a, b for an elliptic curve E: $y^2 + y = x^3 + ax + b$ over $GF(2^m)$; 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 $P_0 = \mathcal{O}$, then output $P_2 \leftarrow P_1$ and stop.
2. If $P_1 = \mathcal{O}$, then output $P_2 \leftarrow P_0$ and stop.
3. If $x_0 \neq x_1$ then
   3.1 Set $\lambda \leftarrow (y_0 + y_1) / (x_0 + x_1)$.
   3.2 Set $x_2 \leftarrow \lambda^2 + x_0 + x_1$.
   3.3 Go to step 7.
4. If $y_0 \neq y_1$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
5. If $x_1 = 0$, then output $P_2 \leftarrow \mathcal{O}$ and stop.
6. Set
   6.1 $\lambda \leftarrow a + x_0^2$.
   6.2 $x_2 \leftarrow \lambda^2$.
7. $y_2 \leftarrow (x_1 + x_2)\lambda + y_1 + 1$.

8. $P_2 \leftarrow (x_2, y_2)$.

The preceding algorithm requires two general multiplications, a squaring, and a multiplicative inversion.

To subtract the point $P = (x, y)$, one adds the point $-P = (x, x + y)$.

A.7.4 Elliptic scalar multiplication

Scalar multiplication can be performed efficiently by the addition–subtraction method outlined as follows.

Input: an integer n and an elliptic curve point P.

Output: the elliptic curve point nP.

1. If $n = 0$, then output $\mathcal{O}$ and stop.
2. If $n < 0$, then set $Q \leftarrow (-P)$ and $k \leftarrow (-n)$, else set $Q \leftarrow P$ and $k \leftarrow n$.
3. Let $h_l h_{l-1} ... h_1 h_0$ be the binary representation of $3k$, where the most significant bit h_l is 1.
4. Let $k_l k_{l-1} ... k_1 k_0$ be the binary representation of k.
5. Set $S \leftarrow Q$.
6. For i from $l - 1$ downto 1 do
   Set $S \leftarrow 2S$.
   If $h_i = 1$ and $k_i = 0$ then compute $S \leftarrow S + Q$ via A.7.1, A.7.2, or A.7.3.
   If $h_i = 0$ and $k_i = 1$ then compute $S \leftarrow S - Q$ via A.7.1, A.7.2, or A.7.3.
7. Output S.

There are several modifications that improve the performance of this algorithm. These methods are summarized in Gordon [B68].

A.7.5 Projective elliptic doubling (prime case)

The projective form of the doubling formula on the curve $y^2 = x^3 + ax + b$ over $GF(p^m)$ for $p > 3$ is

$$2\,(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

where

$$M = 3X_1^2 aZ_1^4$$
$$Z_2 = 2Y_1 Z_1$$
$$S = 4X_1 Y_1^2$$
$$X_2 = M_2 - 2S$$
$$T = 8Y_1^4$$
$$Y_2 = M\,(\mathrm{S} - X_2) - T$$

The algorithm `Double` given as follows performs these calculations.

Input: a modulus p; the coefficients a and b defining a curve E modulo p; projective coordinates (X_1, Y_1, Z_1) for a point P_1 on E.

Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = 2P_1$.

1. $T_1 \leftarrow X_1$.
2. $T_2 \leftarrow Y_1$.

3. $T_3 \leftarrow Z_1$.
4. If $T_2 = 0$ or $T_3 = 0$, then output (1, 1, 0) and stop.
5. If $a = p - 3$, then
   $T_4 \leftarrow T_3^2$.
   $T_5 \leftarrow T_1 - T_4$.
   $T_4 \leftarrow T_1 + T_4$.
   $T_5 \leftarrow T_4 \times T_5$.
   $T_4 \leftarrow 3 \times T_5$.
   $= M$.
   else
   $T_4 \leftarrow a$.
   $T_5 \leftarrow T_3^2$.
   $T_5 \leftarrow T_5^2$.
   $T_5 \leftarrow T_4 \times T_5$.
   $T_4 \leftarrow T_1^2$.
   $T_4 \leftarrow 3 \times T_4$.
   $T_4 \leftarrow T_4 + T_5$.
   $= M$.
6. $T_3 \leftarrow T_2 \times T_3$.
7. $T_3 \leftarrow 2 \times T_3$.
   $= Z_2$.
8. $T_2 \leftarrow T_2^2$.
9. $T_5 \leftarrow T_1 \times T_2$.
10. $T_5 \leftarrow 4 \times T_5$.
    $= S$.
11. $T_1 \leftarrow T_4^2$.
12. $T_1 \leftarrow T_1 - 2 \times T_5$.
    $= X_2$.
13. $T_2 \leftarrow T_2^2$.
14. $T_2 \leftarrow 8 \times T_2$.
    $= T$.
15. $T_5 \leftarrow T_5 - T_1$.
16. $T_5 \leftarrow T_4 \times T_5$.
17. $T_2 \leftarrow T_5 - T_2$.
    $= Y_2$.
18. $X_2 \leftarrow T_1$.
19. $Y_2 \leftarrow T_2$.
20. $Z_2 \leftarrow T_3$.

This algorithm requires 10 field multiplications and 5 temporary variables. If a is small enough that multiplication by a can be done by repeated addition, then only 9 field multiplications are required. If $a = p - 3$, then only 8 field multiplications are required (see Chudnovsky and Chudnovsky [B44]). The proportion of elliptic curves modulo p that can be rescaled so that $a = p - 3$ is about 1/4 if $p \equiv 1 \pmod 4$ and about 1/2 if $p \equiv 3 \pmod 4$. (See A.6.3.1.)

A.7.6 Projective elliptic addition (prime case)

The projective form of the adding formula on the curve $y^2 = x^3 + ax + b$ over $GF(p^m)$ for $p > 3$, is

$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$

where

$U_0 = X_0 Z_1^2$

$S_0 = Y_0 Z_1^3$

$U_1 = X_1 Z_0^2$

$S_1 = Y_1 Z_0^3$

$W = U_0 - U_1$

$R = S_0 - S_1$

$T = U_0 + U_1$

$M = S_0 + S_1$

$Z_2 = Z_0 Z_1 W$

$X_2 = R^2 - TW^2$

$V = TW^2 - 2X_2$

$2Y_2 = VR - MW^3$

The algorithm `Add` given as follows performs these calculations.

Input: a modulus p; the coefficients a and b defining a curve E modulo p; projective coordinates (X_0, Y_0, Z_0) and (X_1, Y_1, Z_1) for points P_0 and P_1 on E, where Z_0 and Z_1 are nonzero.

Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = P_0 + P_1$, unless $P_0 = P_1$. In this case, the triplet (0, 0, 0) is returned. [The triplet (0, 0, 0) is not a valid projective point on the curve, but rather it is a marker indicating that routine `Double` should be used.]

1. $T_1 \leftarrow X_0$ $= U_0$ (if $Z_1 = 1$).
2. $T_2 \leftarrow Y_0$ $= S_0$ (if $Z_1 = 1$).
3. $T_3 \leftarrow Z_0$.
4. $T_4 \leftarrow X_1$.
5. $T_5 \leftarrow Y_1$.
6. If $Z_1 \neq 1$, then
   - $T_6 \leftarrow Z_1$.
   - $T_7 \leftarrow T_6^2$.
   - $T_1 \leftarrow T_1 \times T_7$ $= U_0$ (if $Z_1 \neq 1$).
   - $T_7 \leftarrow T_6 \times T_7$.
   - $T_2 \leftarrow T_2 \times T_7$ $= S_0$ (if $Z_1 \neq 1$).
7. $T_7 \leftarrow T_3^2$.
8. $T_4 \leftarrow T_4 \times T_7$ $= U_1$.
9. $T_7 \leftarrow T_3 \times T_7$.
10. $T_5 \leftarrow T_5 \times T_7$ $= S_1$.
11. $T_4 \leftarrow T_1 - T_4$ $= W$.
12. $T_5 \leftarrow T_2 - T_5$ $= R$.
13. If $T_4 = 0$, then
    - If $T_5 = 0$, then output (0,0,0) and stop.
    - else output (1, 1, 0) and stop.

14. $T_1 \leftarrow 2 \times T_1 - T_4$ $= T.$
15. $T_2 \leftarrow 2 \times T_2 - T_5$ $= M.$
16. If $Z_1 \neq 1$, then
    $T_3 \leftarrow T_3 \times T_6.$
17. $T_3 \leftarrow T_3 \times T_4$ $= Z_2.$
18. $T_7 \leftarrow T_4^2.$
19. $T_4 \leftarrow T_4 \times T_7.$
20. $T_7 \leftarrow T_1 \times T_7.$
21. $T_1 \leftarrow T_5^2.$
22. $T_1 \leftarrow T_1 - T_7$ $= X_2.$
23. $T_7 \leftarrow T_7 - 2 \times T_1$ $= V.$
24. $T_5 \leftarrow T_5 \times T_7.$
25. $T_4 \leftarrow T_2 \times T_4.$
26. $T_2 \leftarrow T_5 - T_4.$
27. $T_2 \leftarrow T_2 / 2$
    $= Y_2.$
28. $X_2 \leftarrow T_1.$
29. $Y_2 \leftarrow T_2.$
30. $Z_2 \leftarrow T_3.$

NOTE—The modular division by 2 in step 27 can be carried out in the same way as in A.2.4. This algorithm can also be used for supersingular ternary curves.

This algorithm requires 16 field multiplications and 7 temporary variables. In the case $Z_1 = 1$, only 11 field multiplications and 6 temporary variables are required. (This is the case of interest for elliptic scalar multiplication.)

A.7.7 Projective elliptic doubling (binary case)

The projective form of the doubling formula on the curve $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^m)$ uses not the coefficient b but rather the field element

$$c := b^{2^{m-2}}$$

computed from b by $m - 2$ squarings. (Thus, $b = c^4$.) The formula is

$$2\,(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

where

$$Z_2 = X_1 Z_1^2$$
$$X_2 = (X_1 + c\,Z_1^2)^4$$
$$U = Z_2 + X_1^2 + Y_1 Z_1$$
$$Y_2 = X_1^4 Z_2 + U X_2$$

The algorithm `Double` given as follows performs these calculations.

Input: a field of 2^m elements; the field elements a and c specifying a curve E over $GF(2^m)$; projective coordinates (X_1, Y_1, Z_1) for a point P_1 on E.

Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = 2P_1$.

1. $T_1 \leftarrow X_1$.
2. $T_2 \leftarrow Y_1$.
3. $T_3 \leftarrow Z_1$.
4. $T_4 \leftarrow c$.
5. If $T_1 = 0$ or $T_3 = 0$ then output (1, 1, 0) and stop.
6. $T_2 \leftarrow T_2 \times T_3$.
7. $T_3 \leftarrow T_3^2$.
8. $T_4 \leftarrow T_3 \times T_4$.
9. $T_3 \leftarrow T_1 \times T_3$ $= Z_2$.
10. $T_2 \leftarrow T_2 + T_3$.
11. $T_4 \leftarrow T_1 + T_4$.
12. $T_4 \leftarrow T_4^2$.
13. $T_4 \leftarrow T_4^2$ $= X_2$.
14. $T_1 \leftarrow T_1^2$.
15. $T_2 \leftarrow T_1 + T_2$ $= U$.
16. $T_2 \leftarrow T_2 \times T_4$.
17. $T_1 \leftarrow T_1^2$.
18. $T_1 \leftarrow T_1 \times T_3$.
19. $T_2 \leftarrow T_1 + T_2$ $= Y_2$.
20. $T_1 \leftarrow T_4$.
21. $X_2 \leftarrow T_1$.
22. $Y_2 \leftarrow T_2$.
23. $Z_2 \leftarrow T_3$.

This algorithm requires five field squarings, five general field multiplications, and four temporary variables.

A.7.8 Projective elliptic addition (binary case)

The projective form of the adding formula on the curve $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^m)$ is as follows:

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

where

$$U_0 = X_0 Z_1^2$$
$$S_0 = Y_0 Z_1^3$$
$$U_1 = X_1 Z_0^2$$
$$W = U_0 + U_1$$
$$S_1 = Y_1 Z_0^3$$
$$R = S_0 + S_1$$
$$L = Z_0 W$$

$V = RX_1 + LY_1$
$Z_2 = LZ_1$
$T = R + Z_2$
$X_2 = a Z_2^2 + TR + W^3$
$Y_2 = TX_2 + VL^2$

The algorithm `Add` given below performs these calculations.

Input: a field of 2^m elements; the field elements a and b defining a curve E over $GF(2^m)$; projective coordinates (X_0, Y_0, Z_0) and (X_1, Y_1, Z_1) for points P_0 and P_1 on E, where Z_0 and Z_1 are nonzero.

Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = P_0 + P_1$, unless $P_0 = P_1$. In this case, the triplet (0, 0, 0) is returned. (The triplet (0, 0, 0) is not a valid projective point on the curve, but rather it is a marker indicating that routine `Double` should be used.)

1. $T_1 \leftarrow X_0$ $= U_0$ (if $Z_1 = 1$).
2. $T_2 \leftarrow Y_0$ $= S_0$ (if $Z_1 = 1$).
3. $T_3 \leftarrow Z_0$.
4. $T_4 \leftarrow X_1$.
5. $T_5 \leftarrow Y_1$.
6. If $a \neq 0$, then
   $T_9 \leftarrow a$.
7. If $Z_1 \neq 1$, then
   $T_6 \leftarrow Z_1$.
   $T_7 \leftarrow T_6^2$.
   $T_1 \leftarrow T_1 \times T_7$ $= U_0$ (if $Z_1 \neq 1$).
   $T_7 \leftarrow T_6 \times T_7$.
   $T_2 \leftarrow T_2 \times T_7$ $= S_0$ (if $Z_1 \neq 1$).
8. $T_7 \leftarrow T_3^2$.
9. $T_8 \leftarrow T_4 \times T_7$ $= U_1$.
10. $T_1 \leftarrow T_1 + T_8$ $= W$.
11. $T_7 \leftarrow T_3 \times T_7$.
12. $T_8 \leftarrow T_5 \times T_7$ $= S_1$.
13. $T_2 \leftarrow T_2 + T_8$ $= R$.
14. If $T_1 = 0$, then
    If $T_2 = 0$, then output (0, 0, 0) and stop.
    else output (1, 1, 0) and stop.
15. $T_4 \leftarrow T_2 \times T_4$.
16. $T_3 \leftarrow T_1 \times T_3$ $= L$ (= Z_2 if $Z_1 = 1$).
17. $T_5 \leftarrow T_3 \times T_5$.
18. $T_4 \leftarrow T_4 + T_5$ $= V$.
19. $T_5 \leftarrow T_3^2$.
20. $T_7 \leftarrow T_4 \times T_5$.
21. If $Z_1 \neq 1$, then
    $T_3 \leftarrow T_3 \times T_6$ $= Z_2$ (if $Z_1 \neq 1$).
22. $T_4 \leftarrow T_2 + T_3$ $= T$.
23. $T_2 \leftarrow T_2 \times T_4$.
24. $T_5 \leftarrow T_1^2$.
25. $T_1 \leftarrow T_1 \times T_5$
26. If $a \neq 0$, then

$T_8 \leftarrow T_3^2$.
$T_9 \leftarrow T_8 \times T_9$.
$T_1 \leftarrow T_1 + T_9$.

27. $T_1 \leftarrow T_1 + T_2$ $= X_2$.
28. $T_4 \leftarrow T_1 \times T_4$.
29. $T_2 \leftarrow T_4 + T_7$ $= Y_2$.
30. $X_2 \leftarrow T_1$.
31. $Y_2 \leftarrow T_2$.
32. $Z_2 \leftarrow T_3$.

This algorithm requires 5 field squarings, 15 general field multiplications, and 9 temporary variables. If $a = 0$, then only 4 field squarings, 14 general field multiplications, and 8 temporary variables are required. [About half of the elliptic curves over $GF(2^m)$ can be rescaled so that $a = 0$. These curves are precisely the curves with order divisible by 4. See A.6.3.1.]

In the case $Z_1 = 1$, only 4 field squarings, 11 general field multiplications, and 8 temporary variables are required. If also $a = 0$, then only 3 field squarings, 10 general field multiplications, and 7 temporary variables are required. (These are the cases of interest for elliptic scalar multiplication.)

A.7.9 Projective full addition and subtraction

The following algorithm `FullAdd` implements a full addition in terms of projective coordinates.

Input: a field of q elements; the field elements a and b defining a curve E over $GF(q)$; projective coordinates (X_0, Y_0, Z_0) and (X_1, Y_1, Z_1) for points P_0 and P_1 on E.

Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = P_0 + P_1$.

1. If $Z_0 = 0$, then output $(X_2, Y_2, Z_2) \leftarrow (X_1, Y_1, Z_1)$ and stop.
2. If $Z_1 = 0$, then output $(X_2, Y_2, Z_2) \leftarrow (X_0, Y_0, Z_0)$ and stop.
3. Set $(X_2, Y_2, Z_2) \leftarrow$ `Add`$[(X_0, Y_0, Z_0), (X_1, Y_1, Z_1)]$.
4. If $(X_2, Y_2, Z_2) = (0, 0, 0)$, then set $(X_2, Y_2, Z_2) \leftarrow$ `Double`$[(X_1, Y_1, Z_1)]$.
5. Output (X_2, Y_2, Z_2).

An *elliptic subtraction* is implemented as follows:

`Subtract`$[(X_0, Y_0, Z_0), (X_1, Y_1, Z_1)] =$ `FullAdd`$[(X_0, Y_0, Z_0), (X_1, U, Z_1)]$

where

$$U = \begin{cases} -Y_1 \bmod p \text{ if } q = p \\ -Y_1 \text{ if } q = 3^m \\ X_1 Z_1 + Y_1 \text{ if } q = 2^m \end{cases}$$

A.7.10 Projective elliptic scalar multiplication

Input: an integer n and an elliptic curve point $P = (X, Y, Z)$.

Output: the elliptic curve point $nP = (X^*, Y^*, Z^*)$.

1. If $n = 0$ or $Z = 0$, then output (1, 1, 0) and stop.
2. Set
   2.1 $X^* \leftarrow X$.
   2.2 $Z^* \leftarrow Z$.
   2.3 $Z_1 \leftarrow 1$.
3. If $n < 0$, then go to step 6.
4. Set
   4.1 $k \leftarrow n$.
   4.2 $Y^* \leftarrow Y$.
5. Go to step 8
6. Set $k \leftarrow (-n)$.
7. If $q = p$, then set $Y^* \leftarrow -Y \pmod p$; else set $Y^* \leftarrow XZ + Y$.
8. If $Z^* = 1$, then set $X_1 \leftarrow X^*$, $Y_1 \leftarrow Y^*$; else set $X_1 \leftarrow X^* / (Z^*)^2$, $Y_1 \leftarrow Y^* / (Z^*)^3$.
9. Let $h_l h_{l-1} \ldots h_1 h_0$ be the binary representation of $3k$, where the most significant bit h_l is 1.
10. Let $k_l k_{l-1} \ldots k_1 k_0$ be the binary representation of k.
11. For i from $l - 1$ down to 1, do
    11.1 Set $(X^*, Y^*, Z^*) \leftarrow$ `Double`$[(X^*, Y^*, Z^*)]$.
    11.2 If $h_i = 1$ and $k_i = 0$, then set $(X^*, Y^*, Z^*) \leftarrow$ `FullAdd`$[(X^*, Y^*, Z^*), (X_1, Y_1, Z_1)]$.
    11.3 If $h_i = 0$ and $k_i = 1$, then set $(X^*, Y^*, Z^*) \leftarrow$ `Subtract`$[(X^*, Y^*, Z^*), (X_1, Y_1, Z_1)]$.
12. Output (X^*, Y^*, Z^*).

There are several modifications that improve the performance of this algorithm. These methods are summarized in Gordon [B68].

A.7.11 Decompression of *y* coordinates (prime case)

The following algorithm recovers the y coordinate of an elliptic curve point from its compressed form.

Input: a prime number p, an elliptic curve E defined over $K = GF(p^m)$ for $p > 3$, the x coordinate of a point (x, y) on E, and the compressed representation $\tilde{y}$ of the y coordinate.

Output: the y coordinate of the point.

1. Compute $g := x^3 + ax + b$ over K.
2. Find a square root z of g modulo p via A.4.4. If the output of A.4.4 is "no square roots exist," then return an error message and stop.
3. Let $\tilde{z}$ be the rightmost bit of z (in other words, z mod 2).
4. If $\tilde{z} = \tilde{y}$, then $y \leftarrow z$, else $y \leftarrow p - z$.
5. Output y.

NOTE—When implementing the algorithm from A.2.5, the existence of modular square roots should be checked. Otherwise, a value may be returned even if no modular square roots exist.

A.7.12 Decompression of *y* coordinates (binary case)

The following algorithm recovers the *y* coordinate of an elliptic curve point from its 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, and the compressed representation $\tilde{y}$ of the y coordinate.

Output: the y coordinate of the point.

1. If $x = 0$, then compute $y - \sqrt{b}$ via A.4.4 and go to step 7.
2. Compute the field element $\alpha := x^3 + ax^2 + b$ in $GF(2^m)$.
3. Compute the element $\beta := \alpha(x^2)^{-1}$ via A.4.2.
4. Find a field element z such that $z^2 + z = \beta$ via A.4.7. If the output of A.4.7 is "no solutions exist," then return an error message and stop.
5. Let $\tilde{z}$ be the rightmost bit of z.
6. Compute $y = (z + \tilde{z} + \tilde{y})x$.
7. Output y.

NOTE—When implementing the algorithm from A.4.7, the existence of solutions to the quadratic equation should be checked. Otherwise, a value may be returned even if no solutions exist.

A.7.13 Decompression of *y* coordinates (ternary case)

The following algorithm recovers the y coordinate of a supersingular elliptic curve point from its compressed form.

Input: a field $GF(3^m)$, a supersingular elliptic curve E defined over $GF(3^m)$, the x coordinate of a point (x, y) on E, and the compressed representation $\tilde{y}$ of the y coordinate.

Output: the y coordinate of the point.

1. Compute $z = x^3 - x - b$ over GF(3^m).
2. Find the square root of z over GF(3^m) via A.4.4 If the output of A.4.4 is "no square root exists," then return an error message and stop.
3. Let $\tilde{z}$ be the leftmost nonzero coefficient of z.
4. If $\tilde{z} = \tilde{y}$, then $y \leftarrow z$ else $y \leftarrow -z$.
5. Output y.

NOTE—When implementing the algorithm from A.4.4, the existence of the quadratic equation should be checked. Otherwise, a value may be returned even if no square root exists.

A.7.14 Finding a random point on an elliptic curve (prime case)

The following algorithm provides an efficient method for finding a random point (other than $\mathcal{O}$) on a given elliptic curve over the finite field $GF(p)$.

Input: a field $K = GF(p^m)$ where $p > 3$ and the parameters a, b of an elliptic curve E over K.

Output: a randomly generated point (other than $\mathcal{O}$) on E.

1. Choose random $x \in K$.
2. Set $\alpha \leftarrow x^3 + ax + b$.
3. If $\alpha = 0$, then output $(x, 0)$ and stop.
4. Apply the appropriate technique from A.4.4 to find a square root modulo p of α or to determine that none exist.
5. If the result of step 4 indicates that no square roots exist, then go to step 1. Otherwise, the output of step 4 is an element β with $0 < \beta < p$ such that

$$\beta^2 \equiv \alpha$$

6. Generate a random bit μ and set $y \leftarrow (-1)^{\mu} \beta$.
7. Output (x, y).

A.7.15 Finding a random point on an elliptic curve (binary case)

The following algorithm provides an efficient method for finding a random point (other than $\mathcal{O}$) on a given elliptic curve over the finite field $GF(2^m)$.

Input: a field $GF(2^m)$ and the parameters *a, b* of an elliptic curve E over $GF(2^m)$.

Output: a randomly generated point (other than $\mathcal{O}$) on E.

1. Choose random x in $GF(2^m)$.
2. If $x = 0$, then output $(0, b^{2^{m-1}})$ and stop.
3. Set $\alpha \leftarrow x^3 + ax^2 + b$.
4. If $\alpha = 0$, then output $(x, 0)$ and stop.
5. Set $\beta \leftarrow x^{-2} \alpha$.
6. Apply the appropriate technique from A.4.7 to find an element z for which $z^2 + z = \beta$ or to determine that none exist.
7. If the result of step 6 indicates that no solutions exist, then go to step 1. Otherwise, the output of step 6 is a solution z.
8. Generate a random bit μ and set $y \leftarrow (z + \mu) x$.
9. Output (x, y).

A.7.16 Finding a random point on an elliptic curve (ternary case)

The following algorithm provides an efficient method for finding a random point (other than $\mathcal{O}$) on a given supersingular elliptic curve over the finite field $GF(3^m)$.

Input: a field $GF(3^m)$ and the parameters *a, b* of an elliptic curve E over $GF(3^m)$.

Output: a randomly generated point (other than $\mathcal{O}$) on E.

1. Choose random y in $GF(3^m)$.
2. Set $\alpha \leftarrow y^2 - b$.
3. If $Tr(\alpha) \neq 0$, then go to step 1.
4. Use A.4.10 to find an element z such that $z^3 - z = \alpha$.
5. Generate a random element $\beta \in GF(3)$.
6. Let $x \leftarrow z + \beta$.
7. Output (x,y).

A.7.17 Finding a point of large prime order

If the order $\#E(GF(q)) = u$ of an elliptic curve E is nearly prime, then the following algorithm efficiently produces a random point on E whose order is the large prime factor r of $u = kr$. (See A.6.3 for the definition of nearly prime.)

Input: a prime r, a positive integer k not divisible by r, and an elliptic curve E over the field $GF(q)$.

Output: if $\#E(GF(q)) = kr$, a point G on E of order r. If not, the message "wrong order."

1. Generate a random point P (not $\mathcal{O}$) on E via A.7.14 or A.7.15.
2. Set $G \leftarrow kP$.
3. If $G = \mathcal{O}$, then go to step 1.
4. Set $Q \leftarrow rG$.
5. If $Q \neq \mathcal{O}$, then output "wrong order" and stop.
6. Output G.

A.7.18 Curve orders over small binary fields

If d is "small" (i.e., it is feasible to perform 2^d arithmetic operations), then the order of the curve $y^2 + xy = x^3 + ax^2 + b$ over $GF(2^d)$ can be calculated directly as follows. Let

$$\mu = (-1)^{\mathrm{Tr}(a)}$$

For each nonzero $x \in GF(2^d)$, let

$$\lambda(x) = \mathrm{Tr}(x + b/x^2)$$

Then

$$\#E(GF(2^d)) = 2^d + 1 + \mu \sum_{x \neq 0} (-1)^{\lambda(x)}$$

A.7.19 Curve orders over extension fields

Given the order of an elliptic curve E over a finite field $GF(2^d)$, the following algorithm computes the order of E over the extension field $GF(2^{de})$.

Input: positive integers d and e, an elliptic curve E defined over $GF(2^d)$, and the order w of E over $GF(2^d)$.

Output: the order u of E over $GF(2^{de})$.

1. Set $P \leftarrow 2^d + 1 - w$ and $Q \leftarrow 2^d$.
2. Compute via A.2.4 the Lucas sequence element V_e.
3. Compute $u := 2^{de} + 1 - V_e$.
4. Output u.

A.7.20 Curve orders via subfields

The algorithms of A.7.18 and A.7.19 allow construction of elliptic curves with known orders over $GF(2^m)$, provided that m is divisible by an integer d that is small enough for A.7.18. The following algorithm finds such curves with nearly prime orders when such exist. (See A.6.3 for the definition of nearly prime.)

Input: a field $GF(2^m)$; a subfield $GF(2^d)$ for some (small) d dividing m; lower and upper bounds $r_{\min}$ and $r_{\max}$ for the base point order.

Output: elements $a, b \in GF(2^m)$ specifying an elliptic curve E, along with the nearly prime order $n = \#E(GF(2^m))$, if one exists; otherwise, the message "no such curve."

1. Select elements $a_0, b_0 \in GF(2^d)$ such that b_0 has not already been selected. (If all of the b_0's have already been tried, then output the message "no such curve" and stop.) Let E be the elliptic curve $y^2 + xy = x^3 + a_0 x^2 + b_0$.
2. Compute the order $w = \#E(GF(2^d))$ via A.7.18.
3. Compute the order $u = \#E(GF(2^m))$ via A.7.19.
4. Test u for near-primality using the techniques in ANSI X9.80-2005 [B11].
5. If u is nearly prime, then set $\lambda \leftarrow 0$ and $n \leftarrow u$ and go to step 9.
6. Set $u' = 2^{m+1} + 2 - u$.
7. Test u' for near-primality using the techniques in ANSI X9.80-2005 [B11].
8. If u' is nearly prime, then set $\lambda \leftarrow 1$ and $n \leftarrow u'$, else go to step 1.
9. Find the elements $a_1, b_1 \in GF(2^m)$ corresponding to a_0 and b_0 via A.5.7.
10. If $\lambda = 0$, then set $\tau \leftarrow 0$. If $\lambda = 1$ and m is odd, then set $\tau \leftarrow 1$. Otherwise, find an element $\tau \in GF(2^m)$ of trace 1 by trial and error using A.4.5.
11. Set $a \leftarrow a_1 + \tau$ and $b \leftarrow b_1$.
12. Output n, a, b.

NOTE—It follows from A.6.3.1 that any a_0 can be chosen at any time in step 1.

A.8 Class group calculations

The following computations are necessary for the complex multiplication technique described in A.9.

A.8.1 Overview

A reduced symmetric matrix is one of the form

$$S = \begin{pmatrix} A & B \\ B & C \end{pmatrix}$$

where the integers A, B, C satisfy the following conditions:

a) GCD $(A, 2B, C) = 1$.

b) $|2B| \le A \le C$.

c) If either $A = |2B|$ or $A = C$, then $B \ge 0$.

The matrix S will be abbreviated as $[A, B, C]$ when typographically convenient.

The determinant $D := AC - B^2$ of S will be assumed throughout this subclause to be positive and squarefree (i.e., containing no square factors).

Given D, the class group $H(D)$ is the set of all reduced symmetric matrices of determinant D. The class number $h(D)$ is the number of matrices in $H(D)$.

The class group is used to construct the reduced class polynomial. This is a polynomial $w_D(t)$ with integer coefficients of degree $h(D)$. The reduced class polynomial is used in A.9 to construct elliptic curves with known orders.

A.8.2 Class group and class number

The following algorithm produces a list of the reduced symmetric matrices of a given determinant D. See Buell [B36].

Input: a squarefree determinant $D > 0$.

Output: the class group $H(D)$.

1. Let s be the largest integer less than $\sqrt{D/3}$.
2. For B from 0 to s do
   - 2.1 List the positive divisors $A_1, \ldots, A_r$ of $D + B^2$ that satisfy $2B \le A \le \sqrt{D+B^2}$.
   - 2.2 For i from 1 to r do
     - 2.2.1 Set $C \leftarrow (D + B^2) / A_i$.
     - 2.2.2 If GCD $(A_i, 2B, C) = 1$, then
       list $[A_i, ,B, C]$.
       if $0 < 2B < A_i < C$, then list $[A_i, -B, C]$.
3. Output list.

Example: $D = 71$. The values of B that need to be checked are $0 \le B < 5$.

- $B = 0$ gives $A = 1$, leading to [1,0,71].
- $B = 1$ gives $A = 2,3,4,6,8$, leading to [3, ±1,24] and [8, ±1,9].
- $B = 2$ gives $A = 5$, leading to [5, ±2, 15].
- $B = 3$ gives $A = 8$, but no reduced matrices.
- $B = 4$ gives no divisors A in the right range.

Thus the class group is

$$H(71) = \{[1,0,71], [3, \pm1,24], [8, \pm1,9], [5, \pm2, 15]\}$$

and the class number is $h(71) = 7$.

A.8.3 Reduced class polynomials

Let

$$F(z) = 1 + \sum_{j=1}^{\infty} (-1)^j \left(z^{\frac{3j^2 - j}{2}} + z^{\frac{3j^2 + j}{2}} \right)$$

$$= 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \ldots$$

and

$$\theta = e^{\left(\frac{-\sqrt{D + Bi}}{A} \pi \right)}$$

Let

$$f_0(A, B, C) = \theta^{-1/24} F(-\theta) / F(\theta^2)$$

$$f_1(A, B, C) = \theta^{-1/24} F(\theta) / F(\theta^2)$$

$$f_2(A, B, C) = \sqrt{2}\ \theta^{1/12} F(\theta^4) / F(\theta^2)$$

NOTE—Because

$$|\theta| < e^{\frac{-\pi\sqrt{3}}{2}} \approx 0.0658287$$

the series $F(z)$ used in computing the numbers $f_J(A, B, C)$ converges as quickly as a power series in $e^{\frac{-\pi\sqrt{3}}{2}}$.

If $[A, B, C]$ is a matrix of determinant D, then its class invariant is

$$\mathrm{C}(A, B, C) = (N \lambda^{BL} 2^{-I/6} (f_J(A, B, C))^K)^G$$

where

$G = \mathrm{GCD}(D, 3)$,

$$I = \begin{cases} 3 & \text{if } D \equiv 1,2,6,7 \pmod 8, \\ 0 & \text{if } D \equiv 3 \pmod 8 \text{ and } D \not\equiv 0 \pmod 3, \\ 2 & \text{if } D \equiv 3 \pmod 8 \text{ and } D \equiv 0 \pmod 3, \\ 6 & \text{if } D \equiv 5 \pmod 8, \end{cases}$$

$$J = \begin{cases} 0 & \text{for } AC \text{ odd,} \\ 1 & \text{for } C \text{ even,} \\ 2 & \text{for } A \text{ even,} \end{cases}$$

$$K = \begin{cases} 2 & \text{if } D \equiv 1,2,6 \pmod 8, \\ 1 & \text{if } D \equiv 3,7 \pmod 8, \\ 4 & \text{if } D \equiv 5 \pmod 8, \end{cases}$$

$$L = \begin{cases} A - C + A^2C & \text{if } AC \text{ odd or } D \equiv 5 \pmod 8 \text{ and } C \text{ even,} \\ A + 2C - AC^2 & \text{if } D \equiv 1,2,3,6,7 \pmod 8 \text{ and } C \text{ even,} \\ A - C + 5AC^2 & \text{if } D \equiv 3 \pmod 8 \text{ and } A \text{ even,} \\ A - C - AC^2 & \text{if } D \equiv 1,2,5,6,7 \pmod 8 \text{ and } A \text{ even,} \end{cases}$$

$$M = \begin{cases} (-1)^{(A^2-1)/8} & \text{if } A \text{ odd,} \\ (-1)^{(C^2-1)/8} & \text{if } A \text{ even,} \end{cases}$$

$$N = \begin{cases} 1 & \text{if } D \equiv 5 \pmod 8 \\ & \quad \text{or } D \equiv 3 \pmod 8 \text{ and } AC \text{ odd} \\ & \quad \text{or } D \equiv 7 \pmod 8 \text{ and } AC \text{ even,} \\ M & \text{if } D \equiv 1,2,6 \pmod 8 \\ & \quad \text{or } D \equiv 7 \pmod 8 \text{ and } AC \text{ odd,} \\ -M & \text{if } D \equiv 3 \pmod 8 \text{ and } AC \text{ even,} \end{cases}$$

$$\lambda = e^{\pi iK/24}.$$

If $[A_1, B_1, C_1]$, ..., $[A_h, B_h, C_h]$ are the reduced symmetric matrices of (positive squarefree) determinant D, then the reduced class polynomial for D is

$$w_D(t) = \prod_{j=1}^{h} \left(t - C\left(A_j, B_j, C_j\right) \right)$$

The reduced class polynomial has integer coefficients.

NOTE—The previous computations need to be performed with sufficient accuracy to identify each coefficient of the polynomial $w_D(t)$. Because each such coefficient is an integer, this means that the error incurred in calculating each coefficient should be less than ½.

Example.

$$w_{71}(t) = \left(t - \frac{1}{\sqrt{2}} \mathsf{f}_0(1,0,71) \right)$$

$$\left(t-\frac{e^{-i\pi/8}}{\sqrt{2}}\mathsf{f}_1(3,1,24)\right)\left(t-\frac{e^{i\pi/8}}{\sqrt{2}}\mathsf{f}_1(3,-1,24)\right)$$

$$\left(t-\frac{e^{-23i\pi/24}}{\sqrt{2}}\mathsf{f}_2(8,1,9)\right)\left(t-\frac{e^{23i\pi/24}}{\sqrt{2}}\mathsf{f}_2(8,-1,9)\right)$$

$$\left(t+\frac{e^{-5i\pi/12}}{\sqrt{2}}\mathsf{f}_0(5,2,15)\right)\left(t+\frac{e^{5i\pi/12}}{\sqrt{2}}\mathsf{f}_0(5,-2,15)\right)$$

$=$ $(t - 2.13060682983889533005591468688942503...)$

$(t - (0.95969178530567025250797047645507504...) +$

$(0.34916071001269654799855316293926907...)\ i)$

$(t - (0.95969178530567025250797047645507504...) -$

$(0.34916071001269654799855316293926907...)\ i)$

$(t + (0.75613568804001789053564010985317 72...) +$

$(0.07375086316308890052407649445676 75...)\ i)$

$(t + (0.75613568804001789053564010985317 72...) -$

$(0.07375086316308890052407649445676 75...)\ i)$

$(t + (0.26885951218510002700028771004661 02...) -$

$(0.84108577401329800103648634224905292...)\ i)$

$(t + (0.26885951218510002700028771004661 02...) +$

$(0. 84108577401329800103648634224905292...)\ i)$

$=$ $t^7 - 2t^6 - t^5 + t^4 + t^3 + t^2 - t - 1.$

A.9 Complex multiplication

A.9.1 Overview

If E is a non-supersingular elliptic curve over $GF(q)$ of order u, then

$$Z = 4q - (q + 1 - u)^2$$

is positive by the Hasse bound (see A.6.3). Thus, there is a unique factorization

$$Z = DV^2$$

where D is squarefree (i.e., contains no square factors). Thus, for each non-supersingular elliptic curve over $GF(q)$ of order u, there exists a unique squarefree positive integer D such that

(*) $4q = W^2 + DV^2$

(**) $u = q + 1 \pm W$

for some W and V.

It is said that E has complex multiplication (CM) by D (or, more properly, by $\sqrt{-D}$). D is called a CM discriminant for q.

If one knows D for a given curve E, then one can compute its order via (*) and (**). As will be demonstrated in the following discussion, one can construct the curves with CM by small D. Therefore, one can obtain curves whose orders u satisfy (*) and (**) for small D. The near-primes are plentiful enough that one can find curves of nearly prime order with small enough D to construct.

Over $GF(p)$, the CM technique is also called the Atkin-Morain method (Morain [B123]); over $GF(2^m)$, it is also called the Lay-Zimmer method (Lay and Zimmer [B103]). Although it is possible [over $GF(p)$] to choose the order first and then the field, it is preferable to choose the field first because there are fields in which the arithmetic is especially efficient.

There are two basic steps involved: finding an appropriate order, and constructing a curve having that order. More precisely, one begins by choosing the field size q, the minimum point order $r_{\min}$, and trial division bound $l_{\max}$. Given those quantities, D is called appropriate if there exists an elliptic curve over $GF(q)$ with CM by D and having nearly prime order.

Step 1 (using A.9.2.3):

Find an appropriate D. When one is found, record D, the large prime r, and the positive integer k such that $u = kr$ is the nearly prime curve order.

Step 2 (using A.9.3):

Given D, k, and r, construct an elliptic curve over $GF(q)$ and a point of order r.

A.9.2 Finding a nearly prime order over *GF*(*p*)

A.9.2.1 Congruence conditions

A squarefree positive integer D can be a CM discriminant for p only if it satisfies the following congruence conditions. Let

$$K \left\lfloor \frac{\left(\sqrt{p}+1\right)^2}{r_{\min}} \right\rfloor$$

- If $p \equiv 3 \pmod 8$, then $D \equiv 2, 3, \text{or } 7 \pmod 8$.
- If $p \equiv 5 \pmod 8$, then D is odd.
- If $p \equiv 7 \pmod 8$, then $D \equiv 3, 6, \text{or } 7 \pmod 8$.
- If $K = 1$, then $D \equiv 3 \pmod 8$.

— If $K = 2$ or 3, then $D \not\equiv 7 \pmod 8$.

Thus, the possible squarefree Ds are as follows:

If $K = 1$, then

D = 3, 11, 19, 35, 43, 51, 59, 67, 83, 91, 107, 115, ….

If $p \equiv 1 \pmod 8$ and $K = 2$ or 3, then

D = 1, 2, 3, 5, 6, 10, 11, 13, 14, 17, 19, 21, ….

If $p \equiv 1 \pmod 8$ and $K \geq 4$, then

D = 1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 17, ….

If $p \equiv 3 \pmod 8$ and $K = 2$ or 3, then

D = 2, 3, 10, 11, 19, 26, 34, 35, 42, 43, 51, 58, ….

If $p \equiv 3 \pmod 8$ and $K \geq 4$, then

D = 2, 3, 7, 10, 11, 15, 19, 23, 26, 31, 34, 35, ….

If $p \equiv 5 \pmod 8$ and $K = 2$ or 3, then

D = 1, 3, 5, 11, 13, 17, 19, 21, 29, 33, 35, 37, ….

If $p \equiv 5 \pmod 8$ and $K \geq 4$, then

D = 1, 3, 5, 7, 11, 13, 15, 17, 19, 21, 23, 29, ….

If $p \equiv 7 \pmod 8$ and $K = 2$ or 3, then

D = 3, 6, 11, 14, 19, 22, 30, 35, 38, 43, 46, 51, ….

If $p \equiv 7 \pmod 8$ and $K \geq 4$, then

D = 3, 6, 7, 11, 14, 15, 19, 22, 23, 30, 31, 35, ….

A.9.2.2 Testing for CM discriminants (prime case)

Input: a prime p and a squarefree positive integer D satisfying the congruence conditions from A.9.2.1.

Output: if D is a CM discriminant for p, then an integer W such that

$$4p = W^2 + DV^2$$

for some V. (In the cases $D = 1$ or 3, the output also includes V.) If not, then the message "not a CM discriminant."

1. Apply the appropriate technique from A.2.5 to find a square root modulo p of $-D$ or to determine that none exist.
2. If the result of step 1 indicates that no square roots exist, then output "not a CM discriminant" and stop. Otherwise, the output of step 1 is an integer B modulo p.
3. Let $A \leftarrow p$ and $C \leftarrow (B^2 + D) / p$.
4. Let $S \leftarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ and $U \leftarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.
5. Until $|2B| \le A \le C$, repeat the following steps:
   5.1 Let $\delta = \left\lfloor \frac{B}{C} + \frac{1}{2} \right\rfloor$.
   5.2 Let $T \leftarrow \begin{pmatrix} 0 & -1 \\ 1 & \delta \end{pmatrix}$.
   5.3 Replace U by $T^{-1}U$.
   5.4 Replace S by $T^t S T$, where T^t denotes the transpose of T.
6. If $D = 11$ and $A = 3$, then let $\delta \leftarrow 0$ and repeat steps 5.2, 5.3, and 5.4.
7. Let X and Y be the entries of U. That is,
$$U = \begin{pmatrix} X \\ Y \end{pmatrix}$$
8. If $D = 1$ or 3, then output $W \leftarrow 2X$ and $V \leftarrow 2Y$ and stop.
9. If $A = 1$, then output $W \leftarrow 2X$ and stop.
10. If $A = 4$, then output $W \leftarrow 4X + BY$ and stop.
11. Output "not a CM discriminant."

A.9.2.3 Finding a nearly prime order (prime case)

Input: a prime p, a trial division bound l_{max}, and lower and upper bounds r_{min} and r_{max} for base point order.

Output: a squarefree positive integer D, a prime r in the interval $r_{min} \le r \le r_{max}$, and a smooth integer k such that $u = kr$ is the order of an elliptic curve modulo p with complex multiplication by D.

1. Choose a squarefree positive integer D, not already chosen, satisfying the congruence conditions of A.9.2.1.
2. Compute via A.2.3 the Jacobi symbol $J = \left(\frac{-D}{p}\right)$. If $J = -1$, then go to step 1.
3. List the odd primes l dividing D.
4. For each l, compute via A.2.3 the Jacobi symbol $J = \left(\frac{p}{l}\right)$. If $J = -1$ for some l, then go to step 1.
5. Test via A.9.2.2 whether D is a CM discriminant for p. If the result is "not a CM discriminant," then go to step 1. (Otherwise, the result is the integer W, along with V if $D = 1$ or 3.)
6. Compile a list of the possible orders, as follows:
   — If $D = 1$, then the orders are
   $$p + 1 \pm W, p + 1 \pm V$$

— If $D = 3$, then the orders are

$$p + 1 \pm W, p + 1 \pm (W + 3V)/2, p + 1 \pm (W - 3V)/2$$

— Otherwise, the orders are $p + 1 \pm W$.

7. Test each order for near-primality. If any order is nearly prime, then output (D, k, r) and stop.
8. Go to step 1.

Example: Let $p = 2^{192} - 2^{64} - 1$. Then

$$p = 4X^2 - 2XY + \frac{1+D}{4} Y^2 \text{ and } p + 1 - (4X - Y) = r$$

where $D = 235$

$$X = -3103725293761793083595768723 4$$

$$Y = 590504615239318452103330511 3$$

and r is the prime

$$r = 627710173538668076383578942333772047398677360825518901532 9$$

Thus, there is a curve modulo p of order r having complex multiplication by D.

A.9.3 Constructing a curve and point (prime case)

A.9.3.1 Constructing a curve with prescribed CM (prime case)

Given a prime p and a CM discriminant D, the following technique produces an elliptic curve $y^2 \equiv x^3 + a_0 x + b_0 \pmod{p}$ with CM by D. (Note that there are at least two possible orders among curves with CM by D. The curve constructed here will have the proper CM, but not necessarily the desired order. This curve will be replaced in A.9.3.2 by one of the desired order.)

For nine values of D, the coefficients of E can be written down at once:

D	a_0	b_0
1	1	0
2	–30	56
3	0	1
7	–35	98
11	–264	1694
19	–152	722
43	–3440	77658
67	–29480	1948226
163	–8697680	9873093538

For other values of D, the following algorithm may be used.

Input: a prime modulus p and a CM discriminant $D > 3$for p.

Output: a_0 and b_0 such that the elliptic curve

$$y^2 \equiv x^3 + a_0 x + b_0 \pmod{p}$$

has CM by D.

1. Compute $w(t) \leftarrow w_D(t) \bmod p$ via A.8.3.
2. Let W be the output from A.9.2.2.
3. If W is even, then use A.5.3 with $d = 1$ to compute a linear factor $t - s$ of $w_D(t)$ modulo p. Let

   $$V := (-1)^D 2^{4I/K} s^{24/(GK)} \bmod p$$

   where G, I and K are as in A.8.3. Finally, let

   $$a_0 := -3(V + 64)(V + 16) \bmod p$$

   $$b_0 := 2(V + 64)^2 (V - 8) \bmod p$$

4. If W is odd, then use A.5.3 with $d = 3$ to find a cubic factor $g(t)$ of $w_D(t)$ modulo p. Perform the following computations, in which the coefficients of the polynomials are integers modulo p.

   $$V(t) := \begin{cases} -t^{24} \bmod g(t) & \text{if } 3 \nmid D \\ -256 t^8 \bmod g(t) & \text{if } 3 \mid D \end{cases}$$

   $$a_1(t) := -3(V(t) + 64)(V(t) + 256) \bmod g(t)$$

   $$b_1(t) := 2(V(t) + 64)^2 (V(t) - 512) \bmod g(t)$$

   $$a_3(t) := a_1(t)^3 \bmod g(t)$$

   $$b_2(t) := b_1(t)^2 \bmod g(t)$$

   Now, let σ be a nonzero coefficient from $a_3(t)$, and let τ be the corresponding coefficient from $b_2(t)$. Finally, let

   $$a_0 := \sigma\tau \bmod p$$

   $$b_0 := \sigma\tau^2 \bmod p$$

5. Output (a_0, b_0).

Example: If $D = 235$, then

$$w_D(t) = t^6 - 10 t^5 + 22 t^4 - 24 t^3 + 16 t^2 - 4 t + 4$$

If $p = 2^{192} - 2^{64} - 1$, then

$$w_D(t) \equiv (t^3 - (5+\phi)t^2 + (1-\phi)t - 2)\,(t^3 - (5-\phi)t^2 + (1+\phi)t - 2) \pmod{p}$$

where ϕ = 1254098248316315745658220082226751383299177953632927607231. The resulting coefficients are

$$a_0 = -208902381629407921389227212 8$$

$$b_0 = -367504956274613540540444576026309668372 48$$

Thus, the curve $y^2 \equiv x^3 + a_0x + b_0$ modulo p has CM by $D = 235$.

A.9.3.2 Choosing the curve and point (prime case)

Input: EC parameters p, k, and r, and coefficients a_0, b_0 produced by A.9.3.

Output: a curve E modulo p and a point G on E of order r, or a message "wrong order."

1. Select an integer ξ with $0 < \xi < p$.
2. If $D = 1$, then set $a \leftarrow a_0\xi \bmod p$ and $b \leftarrow 0$.
   If $D = 3$, then set $a \leftarrow 0$ and $b \leftarrow b_0\xi \bmod p$.
   Otherwise, set $a \leftarrow a_0\xi^2 \bmod p$ and $b \leftarrow b_0\xi^3 \bmod p$.
3. Look for a point G of order r on the curve

   $$y^2 \equiv x^3 + ax + b \pmod{p}$$

   via A.7.17.
4. If the output of A.7.17 is "wrong order," then output the message "wrong order" and stop.
5. Output the coefficients a, b, and the point G.

The method of selecting ξ in the first step of this algorithm depends on the kind of coefficients desired. Two examples follow.

— If $D \neq 1$ or 3, and it is desired that $a = -3$ (see A.7.6), then take ξ to be a solution of the congruence $a_0\xi^2 \equiv -3 \pmod{p}$, provided one exists. If one does not exist, or if this choice of ξ leads to the message "wrong order," then select another curve as follows. If $p \equiv 3 \pmod 4$ and the result was "wrong order," then choose $p - \xi$ in place of ξ; the result leads to a curve with $a = -3$ and the right order. If no solution ξ exists, or if $p \equiv 1 \pmod 4$, then repeat A.9.3 with another root of the reduced class polynomial. The proportion of roots leading to a curve with $a = -3$ and the right order is roughly one half if $p \equiv 3 \pmod 4$, and one quarter if $p \equiv 1 \pmod 4$.

— If there is no restriction on the coefficients, then choose ξ at random. If the output is the message "wrong order," then repeat the algorithm until a set of parameters a, b G is obtained. This will happen for half the values of ξ, unless $D = 1$ (one quarter of the values) or $D = 3$ (one sixth of the values).

A.10 Pairings for cryptography

All pairing algorithms defined here are based on the Miller algorithm (Miller [B121]). We consider only those algorithms that are practical for use in the cryptographic algorithms presented in the main body.

A.10.1 Pairing-friendly elliptic curves

To be useful for pairing-based cryptography, a notion of "pairing-friendly" curves has become established, and this may be formalized by the following conditions. First we define the embedding degree of E with respect to r be defined as the smallest integer k such that $r \mid q^k - 1$. Then an elliptic curve is pairing-friendly if:

There is a prime $r \mid \#E\,(GF(q))$ such that $r > \sqrt{q}$

$k < \log_2(r)\,/8$

A.10.2 Curve families

All the curves we consider are defined over a finite field $K = GF(q)$. For cryptographic use, we also need a suitable subgroup of $E(K)$ of size r. If $\#E(K)$ is the order of the group of K-rational points of E, then the trace t of E/K is $t = q + 1 - \#E(K)$. Note that $r \mid \#E(K)$ and for $h = \#E(K) / r$, we define G_T to be the subgroup of order r *of* $GF(q^h)$.

The search for, and analysis of, pairing-friendly curves has led to a grouping of curves into families, which can often be described by equations in the parameters t, r, and q.

If k is the embedding degree of E, then we let ε be the degree of the maximal twist of E, which we denote E'. Note that $\varepsilon \mid k$ so we define the degree of field over which we consider the twist to be $d = k/\varepsilon$.

Curves may then be classified as follows:

$E_1 = E(GF(q))$

$E_2 = E(GF(q^k))$

$E_3 = E'(GF(q^d))$

Note that $r \mid q^k - 1$ and for E_3, $r \mid \#E'(GF(q))$.

The elements of E_1 of order r form the 1-eigenspace of the Frobenius map with respect to r. This will in all cases be equal to the pairing group G_1. The r-eigenspace of Frobenius lies in E_2.

Pairings may be classified into three different types according to the curve types. In particular, the types depend on how the group G_2 is represented.

In all cases there is a map $\zeta: G_2 \rightarrow E_2$ and the Miller loop is always applied to $\zeta(Q)$ rather than to the element Q itself.

A.10.2.1 Type 1 (*E* supersingular)

$G_1 = G_2$ = a subgroup of E_1 of order r.

A.10.2.2 Type 2 (E ordinary)

G_1 = a subgroup of E_1 of order r.

G_2 = a subgroup of E_2 of order r, which is distinct from both the 1-eigenspace and the q-eigenspace of Frobenius.

A.10.2.3 Type 3 (E ordinary)

G_1 = a subgroup of E_1 of order r.

G_2 = a subgroup of E_3 of order r.

A.10.3 The Miller loop

The Miller loop is a function,

$f_{P,n}(Q)$

where P is on the base curve E_1 and Q is on the extension E_2. In defining the Miller loop, we consider two arbitrary points on the curve and an arbitrary loop length n. When using this loop within a pairing calculation, we will specify more precisely the values of P, Q, and n. There are various optimizations (e.g., for specific curves or by using properties of twisted curves within the algorithm), but only the basic algorithm is given here.

The basic algorithm is as follows:

Input: $P \in E_1, Q \in E_2$

Output: $f_{P,n}(Q)$

Set $T \leftarrow P, f \leftarrow 1$.

1) Write n as $\sum_{i=0}^{m-1} n_i 2^i$ with $n_i \in \{0,1\}$.
2) Loop for i from $m-1$ down to 0
   i. Set $f \leftarrow f^2 l_{T,T}(Q) / v_{2T}(Q)$.
   ii. Set $T \leftarrow 2T$.
   iii. If $n_i = 1$, then
      a. Set $f \leftarrow f.l_{T,P}(Q) / v_{T+P}(Q)$.
      b. Set $T \leftarrow T + P$.
   iv. End if
3) End loop.
4) Return f.
   i. The line functions $l_{A,B}(Q)$ and $v_{A+B}(Q)$ are the functions obtained by substituting the coordinates of Q into the equations for the lines used in the standard formation of $A+B$. l

is the straight line $y - mx - c$, which passes through the points A and B, and v is the vertical line passing through $A+B$.

ii. Hence, if $Q = (x, y)$, $A = (x_1, y_1)$, $B = (x_2, y_2)$, and $A+B = (x_3, y_3)$, then

$$l_{A,B}(Q) = y - \frac{(y_2 - y_1)}{(x_2 - x_1)}x - \frac{(x_1 y_2 - x_2 y_1)}{(x_1 - x_2)}$$

$$v_{A+B}(Q) = x - x_3$$

See A.6.2 for details of how to compute these operations.

A.10.4 Pairing calculations

In the main document, we consider two pairings:

$$e_1: G_1 \times G_2 \rightarrow G_T$$

and

$$e_2: G_2 \times G_1 \rightarrow G_T$$

The pairing e_2 is computed from e_1 via $e_2(P,Q) = e_1(Q,P)$, so from now on, we shall only consider e_1, which we will denote $e(P,Q)$. In all cases, the algorithms and primitives in the main body use pairings with parameters derived from the generators Q_1 of G_1 and Q_2 of G_2. We therefore define the pairings in the follow sublcause with parameters in curves E_1, E_2, or E_3. When the domain parameters are defined, the choice of curve and generators ensures these definitions are consistent.

A.10.5 Pairings

For all of the pairings defined in this subclause, r is the order of G_1, G_2 and P; t is the trace of E over K; and k is the embedding degree of E, all as defined in A.10.1.

A.10.5.1 Tate

The Tate pairing is defined as:

$$e(P,Q) = f_{P,r}(Q)^{(p^k-1)/r}$$

where $P \in E_1$, $Q \in E_2$.

A.10.5.2 Eta

The Eta pairing is only defined for supersingular elliptic curves and is given by the following:

$$e(P,Q) = f_{P,t-1}(Q)^{(p^k-1)/r}$$

where $P \in E_1$, $Q \in E_2$.

A.10.5.3 Ate

The Ate pairing may be computed as follows:

$$e(P,Q) = f_{Q,t-1}(P)^{(p^k-1)/r}$$

where $P \in E_1$, $Q \in E_2$.

A.10.5.4 R-Ate

The R-Ate pairing is defined in terms of two Miller loops and the line and vertical functions from the Miller-Loop definition as follows:

$$e(P,Q) = f_{a,BP}(Q) \cdot f_{b,P}(Q) \cdot l_{aBP,bP}(Q) / v_{aBP+bP}(Q)$$

where $A,B,a,b \in Z$ and $A = aB + b$, $P \in E_1$, $Q \in E_2$.

A.11 Elliptic curves for pairing-based cryptography

A.11.1 Super-singular curves

Supersingular curves have embedding degree $k \in \{1, 2, 3, 4, 6\}$. We only consider those with even embedding degree.

A.11.1.1 Super-singular curves with embedding degree 2

In fields $GF(2^s)$ of characteristic 2, curves with $k = 2$ are of the form

E_1: $y^2 + y = x^3 + \delta x$ where s is even and Tr $\delta \neq 0$ (see A.4.5).

E_2: $y^2 + y = x^3$ where s is odd.

In fields of prime characteristic $q = p > 3$ supersingular curves with $k = 2$ can be defined by:

If $q \equiv 3 \pmod 4$, then $y^2 = x^3 + ax$, where $-a \notin (GF(q))^2 \backslash \{0\}$.

If $q \equiv 5 \pmod 6$, then $y^2 = x^3 + b$.

If $q \equiv 1 \pmod{12}$, then curves may be computed by:

Input: q

Output: An elliptic curve E parameterized in integers m and c

Find D, the smallest prime such that $D \equiv 3 \pmod 4$ and $(-D / q) = -1$

1) Compute a root $j \in GF(q)$ of the reduced class polynomial $H_D \pmod q$ using algorithm A.8.3
2) Set $m = j / (1728 - j)$.
3) Return E: $y^2 = x^3 + 3mc^2x + 2mc^3$ for any c.

A.11.1.2 Super-singular curves with embedding degree 4

There are only two forms of supersingular curve with $k = 4$, and such curves only exist over a field of characteristic 2.

$$E_1: y^2 + y = x^3 + x$$

and

$$E_2: y^2 + y = x^3 + x + 1$$

A.11.1.3 Super-singular curves with embedding degree 6

Supersingular curves with $k = 6$ only exist over fields of characteristic 3, i.e., $GF(3^s)$, and have the form

$$E: y^2 = x^3 - x \pm d$$

where $d \in GF(q)$ with Tr $d = 1$ (see A.4.5).

A.11.2 MNT curves

The Miyaji, Nakabayashi, and Takano (MNT) technique (Miyaji et al. [B122]) for finding ordinary pairing-friendly curves relies on the complex multiplication technique of A.9.3. A resulting curve defined over $GF(p)$ has its order divisible by a large prime r where the r-torsion group of E is defined over an extension field $GF(p^k)$.

Input: $k = 3$, 4, or 6, a maximal cofactor c_{max}, and the maximum discriminant D_{max}.

Output: $GF(q)$, and elliptic curve E such that $|E(GF(q))| = cr$ where $c \le c_{max}$ and r is prime.

1) $\lambda \leftarrow -2\lfloor k/2 \rfloor + 4$.
2) For c from 1 to c_{max} do
    i. For c' from 1 to $4c - 1$, do
        a. $n_k \leftarrow \lambda c + c'$.
        b. $m \leftarrow 4c - c'$.
        c. $f_k \leftarrow n_k^2 - m^2$.
        d. for squarefree $D = 1$ to D_{max} do

1. $s \leftarrow c'mD$.
2. for each solution of $y^2 - sv^2 = f_k$ do
   i. $t \leftarrow (y - n_k)/m + 1$.
   ii. $r \leftarrow \Phi_k(t-1) / c'$.
   iii. $q \leftarrow cr + t - 1$.
   iv. If $t \in \mathbb{Z}$, r is prime and q is prime,
   v. Return q, and E computed by A.9.3, using $p = q$, t, D.

3) Return "fail."

A.11.3 Cocks-Pinch curves

The Cocks-Pinch (CP) approach gives a way to construct pairing-friendly curves with an arbitrary embedding degree. The CP approach works by fixing a prime subgroup size r and a CM discriminant D, and then finding the trace t and prime q such that the CM equation is satisfied.

Input: k, D square free.

Output: $GF(q)$, and elliptic curve E.

1. Choose a prime r such that $k \mid r - 1$ and $\left(\frac{-D}{r}\right) = 1$.
2. Find z a kth root of unity in $(\mathbb{Z}/r\mathbb{Z})^{\times}$.
3. $t' \leftarrow z + 1$.
4. $y' \leftarrow (t'-2)/\sqrt{(-D)} \pmod{r}$.
5. Choose $t \in \mathbb{Z}$ such that $t \equiv t' \pmod{r}'$.
6. Choose $y \in \mathbb{Z}$ such that $y \equiv y' \pmod{r}'$.
7. $q \leftarrow (t^2 + Dy^2)/4$.
8. If q is prime and an integer and $D \leq 10^{12}$, then construct E by A.9.3, using $p = q$, D.

A.11.4 BN curves

Barreto-Naehrig curves are of the form E: $y^2 = x^3 + b$, parameterized by

$r(x) = 36x^4 - 36x^3 + 18x^2 - 6x + 1$

$q(x) = 36x^4 - 36x^3 + 24x^2 - 6x + 1$

$t(x) = 6x^2 + 1$

where r and q are both prime. For such curves, $k = 12$. To find a BN curve, choose random values for x until r and q are both prime then choose $b \in GF(q)$ such that $\#E(GF(p)) = r$.

A.11.5 Kachisa-Schaefer-Scott (KSS) curves

KSS curves are of the form E: $y^2 = x^3+b$, parameterized by the following:

$r(x) = (x^4 – 36x^3 + 16x + 7) / 7$

$q(x) = x^6 + 37x^3 + 343$

$t(x) = (x^8 + 5x^7 + 7x^6 + 37x^5 + 188x^4 + 259x^3 + 343x^2 +1763x +2401) / 21$

where r and q are both prime. For such curves, $k = 18$. To find a KSS curve, choose random values for x until r and q are both prime and then choose $b \in GF(q)$ such that $\#E(GF(p)) = r$.

A.11.6 Brezing-Wang (BW) curves

For $k = 24$, BW curves are of the form E: $y^2 = x^3+b$, parameterized by the following:

$r(x) = x^8 – x^4 + 1$

$q(x) = (x – 1)^2 (x^8 – x^4 + 1)/3 + x$

$t(x) = x + 1$

where r and q are both prime. To find such a BW curve, choose random values for x until r and q are both prime and then choose $b \in GF(q)$ such that $\#E(GF(p)) = r$.

A.12 Choosing a curve and pairing

A.12.1 Security considerations

When considering curves, pairings, and fields, a balance needs to be made between security and efficiency. In general, more efficient arithmetic is obtained if the value $\rho = \log q / \log r$ is small. Hence, to achieve efficiency at a particular security level, it is often necessary to use curves with a large embedding degree k. This is certainly not true in all cases, and efficient arithmetic can be obtained at reasonable security levels with $k = 2$, for example.

Table A.1 lists the minimum recommended sizes of the subgroup and extension fields in terms of bits for different security levels. The security levels are the block cipher key-length equivalents (e.g., AES-128, AES-192, and AES-256 as described in NIST 800-57-2011 [B125]).

Table A.1— Parameter sizes for selected security levels

Security level	$\log_2 r$	$k \times \log_2 q$
112	224	2048
128	256	3072
192	384	8192
256	512	15 360

A.12.2 Curve-pairing compatibility

Not all types of elliptic curves are compatible with the different pairings described in the preceding clauses. Table A.2 summarizes which curve families are compatible with which pairings.

Table A.2—Compatibility of pairings with selected curve types

	Curve family					
Pairing	**Super-singular**	**MNT**	**CP**	**BN**	**KSS**	**BW**
Tate	Yes	Yes	Yes	Yes	Yes	Yes
Eta	Yes	No	No	No	No	No
Ate	No	Yes	Yes	Yes	Yes	Yes
R-Ate	No	Yes	Yes	Yes	Yes	Yes

A.12.3 Suitable domain parameters

Clause A.6 through Clause A.12 have defined various curves and pairings and have presented algorithms for their construction and computation. For each pairing and each curve, specific optimizations of the Miller algorithm can be made to produce efficient computations. Such optimizations have been omitted from this document in the interests of simplicity and practicality. Instead, all pairings are defined in terms of a common, basic Miller loop with $P \in E_1$, $Q' \in E_2$. This means that, in many cases, maps from E_1 to E_2 or E_3 to E_2 are required to form Q' for use in the pairings. Table A.3 summarizes the properties of these mappings for selected types of curves. In practice, when implementing cryptographic schemes using these algorithms, optimized Miller loops may be preferred.

Table A.3—Summary of mappings used to implement pairings using selected curve types

Curve type	Curve	k	Pairing	Order	Map of $Q \rightarrow Q'$
SS	E_1: $y^2 = x^3 + x$ over $GF(p)$, $p \equiv 3 \bmod 4$	2	Tate	$p+1$	$(x,y) \rightarrow (\zeta x, y)$ where ζ is a cube root of unity
SS	E_1: $y^2 = x^3 + x$ over $GF(p)$, $p \equiv 2 \bmod 3$	2	Tate	$p+1$	$(x,y) \rightarrow (x, iy)$ where $i^2 = -1$
SS	E_1: $y^2 + y = x^3 + x + a$, $a = \{1,2\}$ over $GF(2)$	4	Tate	$2^m \pm 2^{(m+1)/2} + 1$	$(x,y) \rightarrow (u^2x + s^2, y + u^2sx + s)$ where $u \in GF(2^2)$, $s \in GF(2^4)$, & $u^2+u+1 = 0$, $s^2+(u+1)s = 0$
SS	E_1: $y^2 = x^3 - x + b$, $b = \pm 1$ over $GF(3)$	6	Tate	$3^m \pm 3^{(m+1)/2} + 1$	$(x,y) \rightarrow (\alpha - x, iy)$ where $i \in GF(3^2)$, $\alpha \in GF(3^3)$, & $i^2 = -1$, $\alpha^3 - \alpha - b = 0$
SS	E_1: $y^2 = x^3 - x + b$, $b = \pm 1$ over $GF(3^m)$	6	Eta	$3^m \pm 3^{(m+1)/2} + 1$	$(x,y) \rightarrow (\rho - x, iy)$ where $i^2 = -1$, $\rho^3 = \rho + b$
SS	E_1: $y^2 + y = x^3 + x + b$, $b \in \{0,1\}$ over $GF(2^m)$	4	Eta	$2^m \pm 2^{(m+1)/2} + 1$	$(x,y) \rightarrow (x - s^2, y + sx + t)$ where $s,t \in$ GF(2^4), $s^2+s+1 = 0$, $t^2 + t + s = 0$
BN	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + b$ over $GF(p^k)$	12	Tate	Parameterized	$Q' = Q$
BN	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + b$ over $GF(p^k)$	12	Ate	Parameterized	$Q' = Q$
BN	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + bv^{-1}$ over $GF(p^k)$ where $3 \mid (p-1)$ & v is a cubic and quadratic nonresidue in $GF(p^2)$	12	Ate	Parameterized	$(x,y) \rightarrow (xv^{1/3}, yv^{1/2})$
MNT	E_1: $y^2 = x^3 + ax + b$ over $GF(p)$ E_2: $y^2 = x^3 + a\,v^{-2}x + bv^{-3}$ over $GF(p^k)$	6	Tate or ate	Parameterized	$(x,y) \rightarrow (vx, v^{3/2}y)$
MNT	E_1: $y^2 = x^3 + ax$ over $GF(p)$ E_2: $y^2 = x^3 + a\,v^{-1}x$ over $GF(p^k)$	6	Tate or ate	Parameterized	$(x,y) \rightarrow (v^{1/2}x, v^{3/4}y)$
MNT	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + bv^{-1}$ over $GF(p^k)$	6	Tate or ate	Parameterized	$(x,y) \rightarrow (v^{1/3}x, v^{1/2}y)$
CP	E_1: $y^2 = x^3 + ax + b$ over $GF(p)$ E_2: $y^2 = x^3 + a\,v^{-2}x + bv^{-3}$ over $GF(p^k)$	Any	Tate or ate	Parameterized	$(x,y) \rightarrow (vx, v^{3/2}y)$
CP	E_1: $y^2 = x^3 + ax$ over $GF(p)$ E_2: $y^2 = x^3 + a\,v^{-1}x$ over $GF(p^k)$	Any	Tate or ate	Parameterized	$(x,y) \rightarrow (v^{1/2}x, v^{3/4}y)$
CP	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + bv^{-1}$ over $GF(p^k)$	Any	Tate or ate	Parameterized	$(x,y) \rightarrow (v^{1/3}x, v^{1/2}y)$
KSS	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + bv^{-1}$ over $GF(p^k)$	18	Tate or ate	Parameterized	$(x,y) \rightarrow (v^{1/3}x, v^{1/2}y)$
BW	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + b$ over $GF(p^k)$	12	Tate	Parameterized	$Q' = Q$
BW	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + b$ over $GF(p^k)$	12	Ate	Parameterized	$Q' = Q$
BW	E_1: $y^2 = x^3 + b$ over $GF(p)$ E_2: $y^2 = x^3 + bv^{-1}$ over $GF(p^k)$ where $3 \mid (p-1)$ & v is a cubic and quadratic nonresidue in $GF(p^2)$	12	Ate	Parameterized	$(x,y) \rightarrow (xv^{1/3}, yv^{1/2})$

A.12.4 Example curves

The following parameters can be used to implement the schemes described in this document. These parameters are meant to be used with the ate pairing as defined in A.10.5.3.

A.12.4.1 Example 1

At the 128-bit security level as defined in NIST 800-57-2011 [B125], writing $\omega = \sqrt{5}$, we have the following:

E_1: $y^2 = x^3 + 5$

E: $y^2 = x^3 + 5$

E_2: $y^2 = x^3 + \omega$

$k = 12$

q = 0xB640D86C60602B112028B881BF7FD34C078201004C25FFFDBFFF550000000001

p = 0xB640D86C60602B112028B881BF7FD34B2F8180C0391C7FFDBFFF550000000001

$t - 1$ = 0xD8008040130980000000000000000000

P_1 = (0x1,
0x18B96D0423CDF2FCEE2CFC51E55988BA58044548921C53F778DD1DFE3BA7CE22)

P_2 =
(0x3E010C633DEB0E3A31B5185D0FDB9A936AF0E26164A830CB5E26B70E6DAFB860 +
0x5B3A6D5A50174E41C25954EE86180B8EF649B35A9F7D95D3D56A9A6B98D0EC9Dω,

0x29700D3295327C272AB24323CF7E2BBFDFE243164692CBDED142729801A9801A +
0x54616AEB636A2296799010D3E4DA893ABBE752502D70B1062313580A88BB452Eω)

A.12.4.2 Example 2

At the 192-bit security level as defined in NIST 800-57-2011 [B125], writing $\omega = \sqrt[3]{2}$, we have the following:

E_1: $y^2 = x^3 + 2$

E: $y^2 = x^3 + 5$

E_2: $y^2 = x^3 + \omega$

$k = 18$

$q =$
0x6B5A6E1D00FF3E57F01A85E93A632A698EBC0675342E193F88B71E95084EF791B2868
06FB1EAFE66EDC2FB8568A06924CC5FBB83EB5F97E1A909DC4AB6194331D

$p =$
0xF42FC00001B36F25E0014387354424803422547260939EE8E63BADEEBAFEAD48BBEF
389BF4EEB7E91CCA352A00000001

$t - 1 =$ 0x6C8700000081047D000039842039A00B6558628750D8C256DCC31700001C88000

$P_1 =$
(0x16E41EDDFEC88FCC60A040689E1D1234A0A805C54CB7FFDFB18893EB3C545C5110
61FDCCA1AE6B686CEF2BB9854DEC2857FC9F1DD0B2280201F5C9D1D5C84425B9,

0x2789E25971C895E57E37A9FFE864A83FE5C8BC4CE589C4C7151A133F8A6E41822C0E
E0F46D25291B9CE8BB0EB15BC22145CD1C5780EFEF6091B4386C3D8F93247A)

$P_2 =$
(0x14984D6199AED6782375E3175C1E25E4D24C4CD3AE6E65430AE14751D6695D1D6A
ABBE570292300B3AD4C176DDCA9A4C8B1747CF098B081787430645C0A8A2A56D +
0x3D0324C329EAC589DE8552005D76A6A7A7638D7165CD392921CFE4BB96C32C2C5D6
D0B8A9147D77119FDA42594DDA0B670366CB76A209346AC0C3310BE3DD4657 ω +
0x4518D02BD8D4CB75DEC343027DC71A513EEBDE1AA232EC7CEBADF2E3F7E1D7AE
4344F5A6717E78001BD5EAD462DC791F360567B251CF2C8E770D185E0794A67194 ω^2,

0x3A6D0FB82C81845322F3880BA86628BFC9716EF028AB41D6F3D107F5858F12DBA557
9146743A8EC57660B22B77CD6D7ADD1A0ECD0DD322738420FA1FFF24BC7958 +
0x227E11E448F0200195795700E03F14F3F2D3412C784E26D1D00F60FF8B6725CCB04964
1F24C8822E73435C3B82762A032359458C56D75C29ED466D7EC277A5BCAA ω +
0x411974A2922DC7804CB921C316B42E203E71DBEBE3BA8DD95B9356025A6F00916C1
8934B0C5A5362C6F5C5A29C6EF5D14DF4BC067A9C261C114E9EFE3EB991AE22 ω^2)

A.12.4.3 Example 3

At the 256-bit security level as defined in NIST 800-57-2011 [B125], writing $\omega_1 = \sqrt{2}$ and $\omega_2 = \sqrt[4]{2}$, we have the following:

E_1: $y^2 = x^3 + 1$

E: $y^2 = x^3 + 5$

E_2: $y^2 = x^3 + 1/\omega_2$

$k = 24$

q =
0x2CC1038ECBEEB01BECF95968F21A7FB9121D72014EA0BDD7B906F8C7D2FB362C5A
12FDACBD5E2C42413D4CF74198E2693978D6FE97ABD52B562FF98DBF0D1EE01DA3C
EDBD7F54F6F2F1EE158774A67B5

p =
0x98C29B81028BC6F535C95626C60D06F9042850D631FCFBBC585F647DE763E467FF0A
D1A96885B7A27118F357E6E6E8E06158E0EFEFFE5F7E6B1984C3BA05F371

$t - 1$ = 0xF000000000800005

P_1 =
(0x647BCE704B0A7F30676B57261DA029E090748839F7DD739AC95F52033FE2C1BDEDF
F6DE9F7F368934A26573B05FC529201ECFF3076E03545F5622550475BCB0B945B9A2D17
D7E08390AC72C659608AE,
0xB82BB0F6F5C80E4C76C78A286AF059440FDFCB15366121F983C323CA63595913EEFF
37CCC7E8933D924F1EA43F561B1838A0B8F0B3CBA1C4B5761E60940EDF6CE251C77C7
117E64C4D2B5FF034B926C)

P_2 =
(0x1FB74F74231E64ADE2E02714FBEE28A84791F927A18FEF5421CDD753DE3E10330C1
FC1E59F3D33550CEF3EBD14042323076FF6E0D3DAE864E028DCFF11C87179E9027DD0
658B44E5F86B9BCB14216807 +
0x1D971D94A7B12E869B0D7997AE345821C791AAE001766216446085479A5CB5459CB0
7D719DFD91DD4689763E57C3A2314AD3022830387AC5F96100BDC1B0EDBD841A7F2C
4E3C2BFE76B9626C501F2A34 ω_1

+
ω_2(0x227A454F6E97C31B9EBCB192B87A6713885218FBE4F332F9C476DC0FB8FC1CAA0
DD73C803A88DCD30E52CC255FFFE526BD6D356D5C54CDAD6B67078B0940C2846D54
7213365D0C6AACDBC64228C444E4 +
0x133EA14E90C5373B20CCE3CB6B5FA14B6C48B6B2B34EDDF645CD02B9D0E88DFB3
A9A2E38F8FFD7C2F1A8C3008B31D4A49A94A40A64D978C46051A523381478588F4D86
78948C97AA25DCEFACBE2FAC64 ω_1),

0x456EFA04B448C5B338306BEE2D62D352B63DF8DB308E9B880504D749868586947CD8
3D1EAEF61860B0225EB5E128A807F2F23188CDFC56F4356C90BAF57082F2DD963ADA
BA7FAD05853D79566F42A07 +
0x2B50BC53A508122622D565EDD8AEE8B9B2E6F5F173D68E065C8DF05732173F9015D
CEDE8F9B7D7898B049E392CA40453BCE2005ED89562AB47290D2EE2BD38D43688AA5
2DEF25C304D809E68C7883DC0 ω_1

+ ω_2 (0x1C6E29D93DF19ACF3B0F6695F4E577C0F47FD2AFFD359056929D5C4D62E55

82E547BA9E33729147346DCE49F76E4DA790B173902403327539FF2B7045B24AEDAA61

0D98F8B96F4947B0E86D06DFCF7BD +
0x2C7C3508D8CB4E0D4FEE1A9DDE53244C4E18192D0A389E8DBFC83AB1014A63E57B
D0D18A88D53C4AA330DA1A2B07E11FE652FBA96C232D4EC14A5727461333614ACB54
BBBB4B3E91D8A15B8E5500AB3D ω_1)

A.12.4.4 Example 4

At the 128-bit security level as defined in NIST 800-57-2011 [B125], we have the following:

E: $y^2 = x^3 - x + 1$

$p = 3$

$m = 509$

$k = 6$

r = 0xF 5538 6BA0 A59D E6BA 9595 87D5 511B D8E8 B716 1C15 392A CB80 DC5F

B128 7455 ABBD B5A4 5FD4 5F94 AEB0 2F9E 80EA 9684 9C66 5DCB FC69

6763 1DF5 DCA1 25B9 0486 4CAA F1CE 8705 F491 78A4 B1CE 5CCC 3274

F4BE BEF6 9ACF 6607 D449 8CDF 5427 8305 2DA7 2BA1 7D0F

$\#E(GF(3^m)) / r$ = 0x7

#

P = (

(1,1,0,1,0,2,2,1,2,0,2,1,1,1,0,2,0,0,0,2,1,1,1,1,2,2,0,1,1,2,0,0,0,0,2,2,0,1,1,

2,2,0,0,2,1,0,1,1,1,0,2,2,1,1,2,0,0,2,1,1,1,2,0,0,0,1,1,2,1,2,0,0,0,0,2,2,1,1,

2,2,2,0,1,2,1,1,1,1,0,1,0,0,0,0,0,0,0,2,0,0,2,1,2,0,2,2,1,0,0,0,1,2,2,0,0,1,0,

2,1,2,1,0,0,1,0,1,1,2,1,1,1,0,1,0,2,2,2,2,2,1,2,1,0,1,0,0,1,1,2,2,0,2,1,1,1,2,2,

1,2,1,0,1,2,0,2,1,0,0,1,1,2,2,1,0,1,2,1,0,2,1,2,0,0,1,0,1,1,1,1,2,0,1,1,2,2,2,

2,2,2,0,0,2,1,2,0,0,1,1,2,2,1,2,0,2,0,2,0,2,1,0,1,0,2,1,0,1,2,0,1,0,2,1,1,1,2,

2,0,2,0,0,1,0,2,1,0,1,2,0,0,2,1,0,2,1,2,1,0,1,0,0,0,0,0,2,2,2,1,1,2,1,0,1,2,1,2,

2,2,0,0,1,1,2,0,2,0,1,1,0,1,0,1,1,1,1,1,2,2,2,0,1,0,0,0,0,0,1,2,1,0,0,1,1,1,0,1,

0,1,1,2,0,2,1,0,0,2,0,0,2,0,2,2,2,0,2,1,0,2,1,0,0,2,1,2,1,1,0,1,0,2,2,2,0,0,0,

2,2,0,0,2,0,0,1,2,0,2,1,0,2,2,2,1,2,1,0,1,0,1,1,0,1,2,0,0,1,0,2,2,2,2,0,0,0,2,2,

1,1,0,1,2,0,2,2,1,0,2,2,2,0,1,2,0,0,2,1,0,1,1,1,0,0,1,1,1,1,2,1,1,1,0,1,2,2,2,0,

1,0,2,0,2,0,2,1,0,1,2,2,1,0,0,2,2,0,0,0,2,0,0,2,1,1,1,1,1,2,2,0,0,2,1,1,0,0,1,

2,1,0,2,2,0,2,2,2,0,1,2,1,0,2,0,0,1,1,1,0,2,1,0,1,1,1,2,0,1,0,1,2,2,0,2,0,0,0,

2,1),

(1,1,1,1,1,2,0,2,2,2,2,2,0,2,0,1,1,1,2,2,0,2,0,1,0,0,1,1,0,2,1,1,0,1,2,0,0,1,0,

1,1,2,0,2,1,2,0,2,0,0,2,0,1,2,1,1,1,1,2,0,2,1,0,2,0,2,1,2,0,1,0,0,2,0,1,1,2,1,

1,1,1,1,2,0,1,2,0,1,2,1,2,1,2,2,0,0,1,2,0,1,1,1,1,0,2,1,1,1,1,0,2,0,2,1,0,0,1,

1,1,0,0,1,2,1,2,2,1,1,2,2,2,1,2,0,1,2,2,1,0,1,2,2,1,0,0,0,0,1,0,0,1,1,1,0,2,2,

1,2,0,0,0,2,0,1,1,0,0,2,2,0,2,1,2,2,2,2,0,2,1,0,1,0,0,1,1,1,1,1,0,2,2,1,1,0,2,0,

2,0,0,2,1,0,1,0,1,1,1,0,0,0,0,0,1,1,1,2,2,1,2,0,2,0,0,0,2,2,2,0,1,1,1,2,1,0,2,

2,1,0,1,2,0,2,0,0,2,1,2,1,1,0,0,0,1,1,0,0,0,1,1,0,2,1,0,2,1,1,2,2,1,2,1,1,1,1,

0,0,0,1,0,1,0,2,0,1,0,2,1,0,0,2,0,2,2,2,1,0,1,1,2,2,2,1,0,1,2,1,1,1,0,0,0,2,1,

0,1,2,0,1,1,1,2,1,2,1,2,0,1,0,2,2,2,2,0,1,1,0,0,0,0,2,1,2,1,0,0,1,0,2,0,1,0,1,0,

1,0,1,2,0,1,0,2,2,2,2,0,1,1,2,2,2,1,0,2,2,0,1,1,2,2,1,0,0,0,0,2,0,1,1,0,2,0,2,

2,2,2,0,2,2,0,1,2,0,2,2,2,1,2,0,2,1,2,0,1,1,1,0,1,2,1,2,1,2,0,1,2,1,0,0,0,1,1,

1,1,2,0,0,2,1,2,1,0,1,0,1,2,2,1,1,0,2,0,1,0,2,1,1,2,1,1,1,0,0,2,1,1,1,1,0,2,0,

2,2,0,2,1,0,0,2,1,2,2,2,2,0,0,2,2,1,1,1,0,0,1,1,0,2,2,2,2,2,0,2,1,1,1,0,1,2,1,2,2,

2,1)

)

Annex B

(normative)

Conformance

The purpose of this annex is to provide implementers with a consistent language for claiming conformance with parts of this standard. Note, however, that this annex does not provide the means for verifying that a particular implementation indeed operates as claimed (this is sometimes called "implementation validation"). Therefore, conformance claims made by an implementation are mere claims unless their accuracy can be assured by other means. Such other means may include, for example, implementation validation or assignment of legal liability to the implementer claiming conformance. Such claims are outside the scope of this standard.

Note also that conformance for the purposes of this standard is a matter of functional correctness, not secure implementation; for the latter, implementers should refer to the security considerations in Annex D.

An implementation may claim conformance with one or more primitives, schemes, or scheme operations specified in this standard, as further described in this annex.

An implementation shall not claim conformance with this standard as a whole.

For background on primitives and schemes, please refer to Clause 4. Specific primitives and schemes are defined in Clause 7 through Clause 11.

B.1 General model

A claim of conformance is an assertion by an implementation that it operates in accordance with some specification over some set of inputs. Thus, a claim of conformance has fundamentally two parts as follows:

— The specification with which conformance is claimed

— A set of inputs, or conformance region, for which the specification is defined, and over which conformance is claimed

For the purposes of this standard, the specification may be that of a primitive, a scheme operation, or a scheme. (An implementation may claim conformance with a scheme by claiming conformance with each operation in the scheme.) For a primitive, the inputs are those stated in the specification; for a scheme operation, the term "input" refers both to initial inputs such as messages and to inputs obtained during a step of the operation such as domain parameters, keys, and key derivation parameters. Recommended conformance regions are given in the specifications.

The set of inputs for which a specification is defined depends on the particular primitive or scheme. For a primitive, the set consists of all inputs that satisfy the input constraints stated for the primitive. For a scheme operation, the set includes at least those inputs that satisfy the input constraints for any primitives invoked by the operation. If the operation includes key validation or domain parameter validation, then the specification may also be defined for certain inputs that do not satisfy the input constraints for a primitive invoked by the scheme. Thus, for example, the specification of a scheme operation may be defined for invalid as well as valid keys when key validation is included in the scheme, even though the specification of a primitive invoked by the scheme is not defined for invalid keys. This is because the behavior of the scheme with key validation is defined as follows on invalid keys: The keys are rejected.

The minimum behavioral requirements for claiming conformance over a conformance region are as follows:

a) On all inputs in the conformance region, the implementation shall perform steps identical to or equivalent to those specified.

b) On all other inputs it accepts, the behavior of the implementation shall not interfere with correct operation on inputs in the conformance region. The behavior is otherwise unconstrained.

Acceptable behaviors in item b) include operating in accordance with the specification (if the specification is defined for the input), rejecting the input, performing steps similar to those specified, or performing some other noninterfering operation.

Because primitives are intended for low-level software or hardware implementation, it may be inconvenient for an implementation of a primitive to check whether an input is supported. Consequently, while an implementation of a primitive may reject some unsupported inputs, it is not expected that an implementation of a primitive will reject every unsupported input. Primitives are not intended to provide security apart from schemes, so such checking is appropriately deferred to the schemes. It is expected that an implementation of a scheme will reject many or even all unsupported inputs, depending on whether key and domain parameter validation is included. For more discussion on the risks of not rejecting unsupported inputs, see D.3.

An implementation may claim conformance over with more than one conformance region, or more than one specification.

NOTE 1—In the interest of interoperability, a conformance region should be sufficiently broad to support a range of possible applications. It is expected that implementation profiles for various applications will give minimum interoperability criteria, in terms of specifications and associated conformance region constraints. For a similar reason, a conformance region should be documented explicitly. (In some cases, however, the documentation may be implicit to some extent; for instance, the domain parameters may be unambiguously specified but secret.)

NOTE 2—Although an implementation's behavior is unconstrained on inputs outside the conformance region (except for not interfering with the behavior on inputs in the conformance region), it is recommended in the interest of robustness that an implementation include checks that prevent failure when specified input constraints are not satisfied. For instance, an implementation should include checks that prevent division by zero, or infinite loops, even if those checks are not necessary when the specified input constraints are satisfied.

NOTE 3—The concept of "equivalence" (as in "perform steps … equivalent to") should be understood in the sense of indistinguishability. A conformant implementation of a scheme or primitive may perform steps identical to those specified for the scheme or primitive, in the sense of performing those steps exactly as specified, or it may perform similar steps that produce the same observable behavior. For instance, if a step calls for generating a random number, then the implementation may generate a pseudorandom number. Under the usual cryptographic assumption that the pseudorandom generator is indistinguishable from a truly random generator, the implementation is equivalent to the specification at that step. Similarly, an implementation may choose to apply restrictions that exclude certain rare events. For instance, an implementation may exclude DL or EC private keys that are equal to 1 and instead generate private keys in the range $[2, r - 1]$. An implementation with such a restriction will be indistinguishable from the specification and may still claim conformance. On the other hand, an implementation that generates private keys in the range [1,1000] could not claim conformance because its behavior would be observably different from the specification. As another example, an implementation of BF might output an error message when the output of the encryption primitive equals its input, which is a rare event.

B.2 Conformance requirements

An implementation claiming conformance with a primitive or scheme specified in this standard shall meet the requirements specified in the clauses of the standard indicated in the following discussion, in addition to the general criteria in B.1. Requirements are to be understood in the context of Clause 2 (Normative

references), Clause 3 (Definitions), Clause 4 (Types of cryptographic techniques), and Clause 5 (Mathematical conventions).

An implementation may claim conformance with a primitive, a scheme, or a scheme operation. For a scheme or scheme operation, conformance requirements for the selected primitive or primitives and for additional techniques such as encoding methods or key derivation functions are also assumed. When documenting conformance with a scheme, these scheme options shall be noted explicitly. In addition, the documentation shall indicate whether the implementation includes key validation or domain parameter validation and, if so, what is validated—i.e., what properties of keys and parameters are assured by the validation. An implementation claiming conformance with a scheme shall satisfy the requirements for each operation in the scheme.

The following is a template for a claim of conformance:

"Conforms with IEEE Std 1363.3-2013 (technique/options) over the region where (constraints on inputs)."

The "technique/options" component identifies the primitive, scheme, or scheme operation; any underlying techniques such as the encoding method or hash function; and any additional choices such as whether and how domain parameter or key validation is performed. The "constraints on inputs" component identifies the conformance region. The method of expressing these components is left to the implementation. Some examples are given in B.3. Table B.1 and Table B.2 list the primitives and schemes for which conformance may be claimed.

Table B.1— Primitives for which conformance can be claimed

Primitive	Subclauses
P-SK-G	7.2.1
P-SK-V	7.2.2
P-SK-E	7.2.3
P-SK-D	7.2.4
P-BB1-G	7.3.2
P-BB1-V	7.3.3
P-BB1-E	7.3.4
P-BB1-D	7.3.5
P-BF-G	7.4.2
P-BF-V	7.4.3
P-BF-E	7.4.4
P-BF-D	7.4.5
P-SCC-D1	7.5.1

Table B.2—Schemes for which conformance can be claimed

Scheme	Operation	Method	Sections
SK KEM	Setup	SK-KEM-S	8.1.1
SK KEM	Extract	SK-KEM-EX	8.1.2
SK KEM	Encapsulate	SK-KEM-EN	8.1.3
SK KEM	Decapsulate	SK-KEM-DE	8.1.4
BB1 KEM	Setup	BB1-KEM-S	8.2.1
BB1 KEM	Extract	BB1-KEM-EX	8.2.2
BB1 KEM	Encapsulate	BB1-KEM-EN	8.2.3
BB1 KEM	Decapsulate	BB1-KEM-DE	8.2.4
BB1 IBE	Setup	BB1-IBE-S	8.3.1
BB1 IBE	Extract	BB1-IBE-EX	8.3.2
BB1 IBE	Encrypt	BB1-IBE-EN	8.3.3
BB1 IBE	Decrypt	BB1-IBE-DE	8.3.4
BF IBE	Setup	BF-IBE-S	8.4.1
BF IBE	Extract	BF-IBE-EX	8.4.2
BF IBE	Encrypt	BF-IBE-EN	8.4.3
BF IBE	Decrypt	BF-IBE-DE	8.4.4
BLMQ Signature	Setup	BLMQ-SIG-S	9.1.2
BLMQ Signature	Extract	BLMQ-SIG-EX	9.1.3
BLMQ Signature	Create signature	BLMQ-SIG-SI	9.1.4
BLMQ Signature	Verify signature	BLMQ-SIG-VE	9.1.5
BLMQ Signcryption	Setup	BLMQ-SC-S	10.1.1
BLMQ Signcryption	Extract	BLMQ-SC-EX	10.1.2
BLMQ Signcryption	Sign and encrypt	BLMQ-SC-SE	10.1.3
BLMQ Signcryption	Decrypt and verify	BLMQ-SC-DV	10.1.4
Wang key agreement	Derive public key	WKA-KA-D1	11.1.1
Wang key agreement	Derive private key	WKA-KA-D2	11.1.2
Wang key agreement	Verification	WKA-KA-V	11.1.3
Wang key agreement	Derive secret value	WKA-KA-D3	11.1.4
Wang key agreement	Generate shared secrets	WKA-KA-G	11.1.5
SCC Key agreement	Generate shared secrets	SCC-KA-G	11.2.1

B.3 Examples

This subclause gives some examples of claims of conformance with the primitives and scheme operations in the standard.

B.3.1 BF IBE

A software module claims conformance with BF IBE. Its conformance claim is as follows:

— *Specification:* BF IBE, as given in 8.4.

— *Conformance region:* Parameters obey the following:

1) G_1 and G_3 are subgroups of order p of $E(GF(q^2))$ where p is a 160-bit prime, q is a 512-bit prime with $q \equiv 2 (\mathrm{mod}\, 3)$, and E is the elliptic curve $E : y^2 = x^3 + 1$.

2) $e_1 : G_1 \times G_1 \rightarrow G_3$ is the reduced, modified Tate pairing.

3) The hash function H_1 is PHFSS.

A module claiming conformance under these conditions may document its conformance as follows:

Conforms with IEEE P136.33-2013 BF IBE over the region where G_1 and G_3 are subgroups of order p of $E(GF(q^2))$ where p is a 160-bit prime, q is a 512-bit prime with $q \equiv 2 (\mathrm{mod}\, 3)$, and E is the elliptic curve $E: y^2 = x^3 + 1$; $e_1 : G_1 \times G_1 \rightarrow G_3$ is the reduced, modified Tate pairing, and the hash function H_1 is PHFSS.

B.3.2 BB_1 KEM

A software module claims conformance with BB_1 KEM. Its conformance claim is as follows:

— *Specification:* BB_1 KEM, as given in 8.2.

— *Conformance region:* Parameters obey the following:

1) G_1 and G_3 are subgroups of order p of $E(GF(q^2))$ where p is a 256-bit prime, q is a 1,536-bit prime with $q \equiv 3 (\mathrm{mod}\, 4)$, and E is the elliptic curve $E: y^2 = x^3 + x$.

2) $e_1 : G_1 \times G_1 \rightarrow G_3$ is the reduced, modified Tate pairing.

A module claiming conformance under these conditions may document its conformance as follows:

Conforms with IEEE P1363.3-2013 BB_1 KEM over the region where G_1 and G_3 are subgroups of order p of $E(GF(q^2))$ where p is a 256-bit prime, q is a 1,536-bit prime with $q \equiv 3 (\mathrm{mod}\, 4)$, and E is the elliptic curve $E: y^2 = x^3 + x$; $e_1 : G_1 \times G_1 \rightarrow G_3$ is the reduced, modified Tate pairing.

Annex C

(informative)

Rationale

This annex is presented in the form of questions and answers. Answers to many general questions that do not deal with identity-based public-key cryptography using pairings can be found in Annex C of IEEE Std 1363-2000.

C.1 General

C.1.1 Why are so many cryptographic techniques defined in this document?

Although not all the techniques defined in this document are established in the marketplace, all of them offer some particular advantage in certain situations. Because it is useful for implementers to have access to a wide range of techniques that can be used to solve a wide range of real-world problems, this document included a wide range of techniques to make this possible.

C.1.2 How were the decisions made regarding the inclusion of individual schemes?

All the schemes that were submitted in the allowed time frame and were deemed to be acceptable to the P1363 Working Group were included in this document.

C.1.3 What is the basis for believing that the schemes defined in this document are secure?

Breaking the schemes defined in this document can be proven to be at least as difficult as solving one of two problems. The first of these is the Computational Bilinear Diffie-Hellman Problem (BDHP). The second in the Computational q-Bilinear Diffie-Hellman Inversion Problem (q-BDHIP).

The setting for both the BDHP and the q-BDHIP is the following. Let G_1, G_2, and G_T be groups and $e: G_1 \times G_2 \rightarrow G_T$ be a bilinear mapping, so that for $g_1 \in G_1, g_2 \in G_2$ and integers a and b we have that $e(g_1^a, g_2) = e(g_1, g_2)^a$ and $e(g_1, g_2^b) = e(g_1, g_2)^b$. Let c be an additional integer.

The BDHP is the following: Given g_i^a, g_j^b, and g_k^c, compute $e(g_1, g_2)^{abc}$, where $(i, j, k) \in \{(1,1,1),(1,1,2),(1,2,2),(2,2,2)\}$. This is assumed to be as difficult as calculating the discrete logs in either G_i, G_j, G_k, or G_T.

The q-BDHIP is the following. Given $g_i^a, g_i^{a^2}, g_i^{a^3}, \ldots, g_i^{a^k}$, compute $e(g_1, g_2)^{1/a}$, where $i \in \{1,2\}$. This is assumed to be as difficult as calculating the discrete logs in G_i.

The relationship between the size of the groups G_1, G_2, and G_T and the security provided by the schemes in this document is the same as that for other public-key schemes based on the intractability of calculating discrete logarithms. More information on the details on the relationship between the size of the groups and the difficulty of calculating discrete logarithms in them can be found in Annex D of IEEE Std 1363-2000.

Annex D

(informative)

Security considerations

D.1 Introduction

In addition to the general comments from Annex D to IEEE Std 1363-2000, the following considerations apply to the techniques for identity-based public-key cryptography that are defined in this document.

D.2 Cryptographic security

Subclause C.1.3 contains more information about the details of the basis for the cryptographic security of the schemes defined in this standard.

D.3 Server secret protection

With the exception of the Wang key agreement scheme defined in 11.1, the cryptographic schemes defined in this document require the use of a server secret that is used to calculate per-user private keys. To ensure the security of the schemes, this server secret needs to be carefully protected because its compromise can potentially allow an adversary to calculate any private key that he or she needs. This is analogous to the requirement of protecting the private key that a certificate authority uses to create digital certificates in X.509-based public-key infrastructure, and server secrets should be protected with the same level of security as that of the private key of a certificate authority.

Annex E

(informative)

Formats

E.1 Overview

As outlined in Clause 4, the specifications presented in this standard are functional specifications rather than interface specifications. Therefore, this standard does not specify how the mathematical and cryptographic objects (such as field elements, keys, or outputs of schemes) are to be represented for the purposes of communication or storage. This informative annex provides references to other relevant standards and defines some recommended primitives for that purpose. Although the use of this annex is optional, it is recommended for interoperability.

As octet strings are arguably the most common way to represent data electronically for the purposes of communication, this annex focuses on representing objects as octet strings. One way to accomplish this is to represent data structures in Abstract Syntax Notation 1 (ASN.1—see ISO/IEC 8824-1:2002 [B77], ISO/IEC 8824-2:2002 [B78], ISO/IEC 8824-3:2002 [B79], and ISO/IEC 8824-4:2002 [B80]) and then to use encoding rules, such as Basic Encoding Rules, Distinguished Encoding Rules, or others (see ISO/IEC 8825-1:2002 [B81] and ISO/IEC 8825-2:2002 [B82]) to represent them as octet strings. This annex does not specify ASN.1 constructs for use in this standard because the generality of this standard would make such constructs very complex. It is likely that particular implementations only use a small part of the options available in this standard and would be better served by simpler ASN.1 constructs. When the use of ASN.1 is desired, ASN.1 constructs defined in the following standards or draft standards may be adapted for use:

- ANSI X9.42-2003 [B5] for DL key agreement
- ANSI X9.63-2001 [B10] for EC key agreement and EC encryption for key transport
- ANSI X9.57-1997 [B8] for DL DSA signatures
- ANSI X9.62-2005 [B9] for EC DSA signatures
- ANSI X9.31-1998 [B4] for IF signatures
- ANSI X9.44-2007 [B6] for IF encryption for key transport

Additional examples of ASN.1 syntax can be found in documents such as PKCS #1 [B133] and SEC1 (Bleichenbacher [B23]). Alternatives to ASN.1 syntax are also available. In the digital signature specification for the eXtensible Markup Language (XML) (Solo et al. [B152]), public keys and digital signatures are represented as ASCII text strings, whereas in Transport Layer Security (TLS) (Dierks and Rescorla [B52]), various cryptographic values are represented as octet strings with a syntax that is similar to data structures in the C programming language.

Subclause E.2 gives recommendations on representing basic mathematical objects as octet strings.

E.2 Representing basic data types as octet strings

When integers, finite field elements, elliptic curve points, or binary polynomials need to be represented as octet strings, it should be done as described in this subclause. Other primitives for converting between different data types (including bit strings) are defined in Clause 5.

E.2.1 Integers (I2OSP and OS2IP)

Integers should be converted to/from octet strings using primitives I2OSP and OS2IP, as defined in 5.6.3.

E.2.2 Finite field elements (FE2OSP and OS2FEP)

Finite field elements should be converted to/from octet strings using primitives FE2OSP and OS2FEP, as defined in 5.6.4.

E.2.3 Elliptic curve points (EC2OSP and OS2ECP)

Elliptic curve points should be converted to/from octet strings using one of the pairs of primitives defined in 5.6.6.2 or 5.6.6.3. The *x*-coordinate-only representation in 5.6.6.3 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.

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.7. See A.7 also for more information on point compression.) The uncompressed form of P is simply given by its two coordinates. The compressed form is presented in E.2.4. The octet string format is defined to support both compressed and uncompressed points.

E.2.4 Polynomials over *GF*(*p*), *p* ≥ 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, defined in the subsequent paragraph.

If $p = 2$, then 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 5.6.4). If $p \geq 3$, then 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 $p = 2$, then to represent $f(t)$ as an octet string, the bits representing its coefficients should be concatenated into a single bit string: $a = a_e \,||\, a_{e-1} \,||\, \ldots \,||\, a_1 \,||\, a_0$. The bit string a should then be converted into an octet string using BS2OSP (see 5.6.2). If $p \geq 3$, then the integer

$$a = a_{t-1}p^{t-1} + \ldots + a_2p^2 + a_1p + a_0$$

should first be computed and then converted into an octet string using I2OSP (see 5.6.3).

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 $p = 2$, then OS2PNP should use OS2BSP (see 5.6.2) 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, one bit each. Note that at most seven leftmost coefficients may be zero. The leftmost zero coefficients should be discarded.

If $p \geq 3$, then OS2PNP should use OS2IP (see 5.6.3) to obtain the integer a defined previously. 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 in this subclause is intended for arbitrary polynomials. More compact representations are possible for sparse polynomials; see Schroeppel and Eastlake [B144] for an example.

Annex F

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] ANSI X9.17-1995, Financial Institution Key Management (Wholesale).

[B2] ANSI X9.30:1-1997, Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA) (revision of X9.30:1-1995).

[B3] ANSI X9.30:2-1997, Public Key Cryptography for the Financial Services Industry: Part 2: The Secure Hash Algorithm (SHA-1) (revision of X9.30:2-1993).

[B4] ANSI X9.31-1998, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA).

[B5] ANSI X9.42-2003, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Diffie-Hellman and MQV Algorithms.

[B6] ANSI X9.44-2007, Key Establishment Using Integer Factorization Cryptography.

[B7] ANSI X9.52-1998, Cryptography for the Financial Services Industry: Triple Data Encryption Algorithm Modes of Operation.

[B8] ANSI X9.57-1997, Public Key Cryptography for the Financial Services Industry: Certificate Management.

[B9] ANSI X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[B10] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Transport using Elliptic Curve Cryptography.

[B11] ANSI X9.80-2005, Prime Number Generation, Primality Testing and Primalilty Certificates.

[B12] ANSI X9.TG-17, 1998, Public-Key Cryptography for the Financial Services Industry: Technical Guideline on Elliptic Curve Arithmetic.

[B13] Adams, C., and M. Myers, "Certificate management message formats," Internet Engineering Task Force (IETF), PKIX working group, work in progress.[6]

[B14] Anderson, R., and M. Kuhn, "Soft tempest: Hidden data transmission using electromagnetic emanations," D. Aucsmith, Ed., *Second International Workshop on Information Hiding—IH'98, Lecture Notes in Computer Science,* vol. 1525. New York: Springer-Verlag, 1998.

[B15] Ash, D. et al., "Low complexity normal bases," *Discrete Applied Mathematics,* vol. 25, pp. 191-210, 1989.

[B16] Atkin, O., "Square roots and cognate matters modulo $p = 8n + 5$," Internet communication to Number Theory mailing list (11 Nov. 1992).[7]

[6] Available at http://www.ietf.org/ids.by.wg/pkix.html.
[7] Archived at http://listserv.nodak.edu/scripts/wa.exe?A2=ind9211&L=nmbrthry&O=T&P=562.

[B17] Bellare, M., and P. Rogaway, "Optimal asymmetric encryption—how to encrypt with RSA," A. De Santis, Ed., *Advances in Cryptology—EUROCRYPT '94, Lecture Notes in Computer Science*, vol. 950. New York: Springer-Verlag, 1995, pp. 92–111.[8]

[B18] Bellare, M., and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," U. M. Maurer, Ed., *Advances in Cryptology—EUROCRYPT '96, Lecture Notes in Computer Science,* vol. 1070. New York: Springer-Verlag, 1996, pp. 399–416.[9]

[B19] Bellare, M. et al., "Relations among notions of security for public-key encryption schemes," H. Krawczyk, Ed., *Advances in Cryptology—CRYPTO '98, Lecture Notes in Computer Science*, vol. 1462. New York: Springer-Verlag, 1998, pp. 26–45.[10]

[B20] Berlekamp, E., *Algebraic Coding Theory*. New York: McGraw-Hill, 1968, pp. 36–44.

[B21] Blake-Wilson, S. et al., "Key agreement protocols and their security analysis," M. Darnell, Ed., *Cryptography and Coding: Sixth IMA International Conf., Lecture Notes in Computer Science*, vol. 1355. New York: Springer-Verlag, 1997, pp. 30–45.[11]

[B22] Blake-Wilson, S., and A. Menezes, "Unknown key-share attacks on the station-to-station (STS) protocol," H. Imai and Y. Zheng, Eds., *Public Key Cryptography: Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99, Lecture Notes in Computer Science,* vol. 1560, pp. 154–170, 1999.[12]

[B23] Bleichenbacher, D., "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," H. Krawczyk, Ed., *Advances in Cryptology—CRYPTO '98*, *Lecture Notes in Computer Science,* vol. 1462. New York: Springer-Verlag, 1998, pp. 1–12.

[B24] Blum, L. et al., "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing,* vol. 15, pp. 364–383, 1986.

[B25] Blum, M., and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing,* vol. 13, pp. 850–864, 1984.

[B26] Boneh, D., and X. Boyen, "Efficient selective-ID secure identity based encryption without random oracles," *Advances in Cryptology—EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2004, pp. 223–238.

[B27] Boneh, D., and G. Durfee, "Cryptoanalysis of RSA with private key d less than $N^{0.292}$," J. Stern, Ed., *Advances in Cryptology—EUROCRYPT '99, Lecture Notes in Computer Science,* vol. 1592. Berlin: Springer-Verlag, 1999, pp. 1–11.

[B28] Boneh, D., and M. Franklin, "Identity based encryption from the Weil pairing," *Advances in Cryptology - Crypto 2001*, vol. 2139 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2001, pp. 213–229.

[B29] Boneh, D., and R. Lipton, "Algorithms for black box fields and their application to cryptography," N. Koblitz, Ed., *Advances in Cryptology—CRYPTO '96*, *Lecture Notes in Computer Science,* vol. 1109. Berlin: Springer-Verlag, 1996, pp. 283–297.

[B30] Boneh, D., and R. Venkatesan, "Breaking RSA may not be equivalent to factoring," K. Nyberg, Ed., *Advances in Cryptology—EUROCRYPT '98, Lecture Notes in Computer Science,* vol. 1403. Berlin: Springer-Verlag, 1998, pp. 59–71.

[B31] Boneh, D. et al., "An attack on RSA given a small fraction of the private key bits," K. Ohta and D. Pei, Eds*., Advances in Cryptology – ASIACRYPT '98, Lecture Notes In Computer Science,* vol. 1514. Berlin: Springer-Verlag, 1998, pp. 25–34.

[8] Revised version appears in http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html.
[9] Revised version appears in http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html.
[10] Full version appears in http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html.
[11] A full version is available from http://www.cacr.math.uwaterloo.ca/.
[12] Also available as technical report CORR 98-42 from http://www.cacr.math.uwaterloo.ca/.

[B32] Boneh, D. et al., "On the importance of checking cryptographic protocols for faults," W. Fumy, Ed., *Advances in Cryptology—EUROCRYPT '97, Lecture Notes in Computer Science,* vol. 1223. Berlin: Springer-Verlag, 1997, pp. 37–51.

[B33] Boyen, X., "A promenade through the new cryptography of bilinear pairings," *Proc. Information Theory Workshop—2006*, Punta del Este, Uruguay, pp. 19–23, 2006.

[B34] Brillhart, J. et al., *Factorizations of $b^n \pm 1$, b = 2,3, 5,6,7,10,11,12, up to High Powers*, 2d ed. Providence, RI: American Mathematical Society, 1988.

[B35] Buchmann, J., et al., "An implementation of the general number field sieve," D. Stinson, Ed., *Advances in Cryptology—CRYPTO '93, Lecture Notes in Computer Science*, vol. **77**3. Berlin: Springer-Verlag, 1994, pp. 159–165.

[B36] Buell, D., *Binary Quadratic Forms: Classical Theory and Modern Computations*. Berlin: Springer-Verlag, 1989.

[B37] Buhler, J. et al., "Factoring integers with the number field sieve," A. K. Lenstra and H.W. Lenstra, Jr., Eds., *The Development of the Number Field Sieve, Lecture Notes in Mathematics,* vol. 1554. Berlin: Springer-Verlag, 1993, pp. 50–94.

[B38] Burthe, R., Jr., "Further investigations with the strong probable prime test," *Mathematics of Computation,* vol. 65, pp. 373–381, 1996.

[B39] Chen, L. et al., "An efficient ID-KEM based on the Sakai-Kasahara key construction," *IEE Proc., Information Security*, vol. 153, pp. 19–26, 2006.

[B40] Chen, L., and C. Williams, "Public Key Sterilization," unpublished draft, Aug. 1998.

[B41] Chen, L., and Z. Cheng, "Security proof of Sakai-Kasahara's identity-based encryption scheme," N. Smart, Ed., *Proc. of the 10th IMA International Conf. on Cryptography and Coding*, LNCS 3796. Berlin: Springer-Verlag, 2005, pp. 442–459.

[B42] Chen, L. et al., "Identity-based key agreement protocols from pairings," *Int. Journal of Information Security*, vol. 6. Berlin: Springer, 2007, pp. 213–241.

[B43] Chen, M., and E. Hughes, "Protocol Failures Related to Order of Encryption and Signature: Computation of Discrete Logarithms in RSA Groups," C. Boyd and E. Dawson, Eds., *Third Australian Conf. on Information Security and Privacy—ACISP '98, Lecture Notes in Computer Science,* vol. 1438, 1998.

[B44] Chudnovsky, D. V., and G.V. Chudnovsky, "Sequences of numbers generated by addition in formal groups and new primality and factorizations tests," *Advances in Applied Mathematics*, vol. **7**, pp. 385–434, 1987.

[B45] Coppersmith, D. et al., "ISO 9796-1 and the new forgery strategy (working draft)." Presented at the rump session of *CRYPTO '99*.[13]

[B46] Coppersmith, D. et al., "Low-exponent RSA with related messages," U. M. Maurer, Ed., *Advances in Cryptology—EUROCRYPT '96, Lecture Notes in Computer Science,* vol. 1070. Berlin: Springer-Verlag, 1996, pp. 1–9.

[B47] Coron, J., and D. Naccache, "An accurate evaluation of Maurer's universal test," *Selected Areas in Cryptography—SAC '98, Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1998.

[B48] Coron, J., et al., "On the security of RSA padding," M. J. Wiener, Ed., *Advances in Cryptology—CRYPTO '99, Lecture Notes in Computer Science,* vol. 1666. Berlin: Springer-Verlag, 1999, pp. 1–18.

[B49] Damgard, I. et al., "Average case error estimates for the strong probable prime test," *Mathematics of Computation,* vol. 61, pp. 177–194, 1993.

[13] Available from http://grouper.ieee.org/groups/1363/contrib.html.

[B50] Davis, D. et al., "Cryptographic randomness from air turbulence in disk drives," Y. Desmedt, Ed., *Advances in Cryptology—CRYPTO '94, Lecture Notes in Computer Science,* vol. 839. Berlin: Springer-Verlag, 1994, pp. 114–120.

[B51] Dhem, J. et al., "A Practical Implementation of the Timing Attack," *CARDIS '98*, *Lecture Notes in Computer Science,* Springer Verlag, 1998.

[B52] Dierks, T., and E. Rescorla, "The Transport Level Security (TLS) Protocol Version 1.2," August 2008, RFC 5246.[14]

[B53] Diffie, W., "The first ten years of public-key cryptology," *Proc. of the IEEE,* vol. 76, pp. 560–577, 1988.

[B54] Diffie, W., and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory* vol. 22, pp. 644–654, 1976.

[B55] Diffie, W. et al., "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography,* vol. 2, pp. 107–125, 1992.

[B56] Dobbertin, H. et al., "RIPEMD-160: A strengthened version of RIPEMD," D. Gollmann, Ed., *Fast Software Encryption, Third Int. Workshop, Lecture Notes in Computer Science,* vol. 1039. Berlin: Springer-Verlag, 1996, pp. 71–82.[15]

[B57] Dodson, B., and A. Lenstra, "NFS with four large primes: An explosive experiment," D. Coppersmith, Ed., *Advances in Cryptology—CRYPTO '95, Lecture Notes in Computer Science,* vol. 963. Berlin: Springer-Verlag, 1995, pp. 372–385.

[B58] Federal Information Processing Standards Publication 140-1, Security Requirements for Cryptographic Modules, April 11, 1994 (supersedes FIPS PUB 140).[16]

[B59] Federal Information Processing Standards Publication 140-2, Security Requirements for Cryptographic Modules, May 25, 2001 (supersedes FIPS PUB 140-1).[17]

[B60] Federal Information Processing Standards Publication 180-4, Secure Hash Standard, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, Virginia, March, 2012.[18]

[B61] Federal Information Processing Standards Publication 186, Digital Signature Standard, 1994.[19]

[B62] Gallant, R. et al., "Improving the parallelized Pollard lambda search on binary anomalous curves," *Mathematics of Computation*, to appear.

[B63] Gennaro, R. et al., "An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products," *Proc. of the 5th ACM Conf. on Computer and Communications Security (CCS-5)*, 1998, pp. 67–72.[20]

[B64] Gilbert, H. et al., "Attacks on Shamir's 'RSA for paranoids.'" *Information Processing Letters,* vol. 68, pp.197–199, 1998.[21]

[B65] Goldwasser, S., and J. Kilian, "Almost all primes can be quickly certified," *Proc. of the 18th Annu. ACM Symp. on Theory of Computing,* 1986, pp. 316–329.

[B66] Goldwasser, S., and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, pp. 270–299, 1984.

[14] Available at http://tools.ietf.org/html/rfc5246.
[15] A corrected and updated version is available from http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html.
[16] Available at http://www.itl.nist.gov/div897/pubs/fip140-1.htm.
[17] Available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.
[18] Available at http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.
[19] Available at http://www.itl.nist.gov/div897/pubs/fip186.htm.
[20] Available from http://www.acm.org/pubs/articles/proceedings/commsec/288090/p67-gennaro/p67-gennaro.pdf.
[21] Also available from http://www.research.att.com/~amo/doc/crypto.html.

[B67] Goldwasser, S. et al., "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing,* vol. 17, pp. 281–308, 1988.

[B68] Gordon, D., "A survey of fast exponentiation methods," *Journal of Algorithms,* vol. 27, pp. 129–146, 1998.

[B69] Gordon, D., "Designing and detecting trapdoors for discrete log cryptosystems," E. Brickell, Ed., *Advances in Cryptology—CRYPTO '92, Lecture Notes in Computer Science,* vol. 740. Berlin: Springer-Verlag, 1993, pp. 66–75.

[B70] Gordon, D., "Discrete logarithms in *GF* (*p*) using the number field sieve," *SIAM Journal on Discrete Mathematics*, vol. 6, pp. 124–138, 1993.

[B71] Gordon, D., and K. McCurley, "Massively parallel computations of discrete logarithms," E. Brickell, Ed, *Advances in Cryptology—CRYPTO '92, Lecture Notes in Computer Science,* vol. 740. Berlin: Springer-Verlag, 1993, pp. 312–323.

[B72] Goss, K., "Cryptographic Method and Apparatus for Public Key Exchange with Authentications," U.S. Patent no. 4,956,863, Sept. 11, 1990.

[B73] Guillou, C. et al., "Precautions taken against various potential attacks in ISO/IEC DIS 9796," I.B. Damgard, Ed., *Advances in Cryptology—EUROCRYPT '90, Lecture Notes in Computer Science,* vol. 473. Berlin: Springer-Verlag, 1991, pp. 465–473.

[B74] Gunther, C., "An identity-based key-exchange protocol," J. Quisquater and J. Vandewalle, Eds., *Advances in Cryptology—EUROCRYPT '89, Lecture Notes in Computer Science,* vol. 434. Berlin: Springer-Verlag, 1990, pp. 29–37.

[B75] Hafner, K., and J. Markoff, *Cyberpunk: Outlaws and Hackers on the Computer Frontier*, updated edition. New York: Touchstone Books, 1995.

[B76] Hastad, J., "Solving simultaneous modular equations of low degree," *SIAM Journal on Computing,* vol. 17, pp. 336–341, 1988.

[B77] ISO/IEC 8824-1:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. Equivalent to ITU-T Rec. X.680 (2002).[22]

[B78] ISO/IEC 8824-2:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Information Object Specification. Equivalent to ITU-T Rec. X.681 (2002).

[B79] ISO/IEC 8824-3:2002, Information Technology—Abstract Syntax Notation One (ASN.1): Constraint Specification. Equivalent to ITU-T Rec. X.682 (2002).

[B80] 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 (2002).

[B81] 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 (DER). Equivalent to ITU-T Rec. X.690 (2002).

[B82] ISO/IEC 8825-2:2002, Information Technology—ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER). Equivalent to ITU-T Rec. X.691 (2002).

[B83] ISO/IEC 9796:1991, Information Technology—Security Techniques—Digital Signature Scheme Giving Message Recovery.

[B84] ISO/IEC 9796-4-1998, Information Technology—Security Techniques—Digital Signature Schemes Giving Message Recovery—Part 4: Methods Based on the Discrete Logarithm.

[B85] ISO/IEC DIS 14888-3-1998, Information Technology—Security Techniques—Digital Signature with Appendix—Part 3: Certificate-Based Mechanisms.

[22] ISO/IEC publications are available from the ISO Central Secretariat (http://www.iso.org/). ISO publications are also available in the United States from the American National Standards Institute (http://www.ansi.org/).

[B86] Itoh, T., et al., "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^t)$ using normal bases," *Journal of the Society for Electronic Communications (Japan),* vol. 44, pp. 31–36, 1986.

[B87] ITU-T Recommendation X.509 (E 29041)-2005, Information Technology—Open Systems Interconnection—The Directory: Public-key and Attribute Certificate Frameworks.[23]

[B88] Johnson, D., unpublished communication to ANSI X9F1 and IEEE P1363 working groups.

[B89] Johnson, D., and S. Matyas, "Asymmetric encryption: Evolution and enhancements," *CryptoBytes* vol. 2 no. 1 (Spring 1996), RSA Laboratories.[24]

[B90] Joye, M., and J. Quisquater, "Efficient computation of full Lucas sequences," *Electronics Letters,* vol. 32, pp. 537–538, 1996.[25]

[B91] Joye, M., and J. Quisquater, "On Rabin-type signatures (working draft)." Presented at the rump session of *CRYPTO '99*.[26]

[B92] Kaliski, B., Jr., "Compatible cofactor multiplication for Diffie-Hellman primitives," *Electronics Letters*, vol. 34, pp. 2396–2397, 1998.

[B93] Kaliski, B., Jr., "MQV vulnerability," Internet communication to ANSI X9F1 and IEEE P1363 mailing lists, June 17, 1998.

[B94] Kehoe, B., *Zen and the Art of the Internet: A Beginner's Guide*, 4th ed. Upper Saddle River, NJ: Prentice Hall Computer Books, 1995.

[B95] Kelsey, J. et al., "Cryptanalytic attacks on pseudorandom number generators," S. Vaudenay, Ed., *Fast Software Encryption, Fifth International Workshop Proc., Lecture Notes in Computer Science,* vol. 1372. Berlin: Springer-Verlag, 1998, pp. 168–188.

[B96] Kerckhoffs, A., "La cryptographie militaire," *Journal des Sciences Militaires*, 9th Series, pp. 161–191, 1883.

[B97] Knuth, D., *The Art of Computer Programming*. *Vol. 2: Seminumerical Algorithms*, 2d ed. Boston, MA: Addison-Wesley, 1981, p. 379.

[B98] Koblitz, N., *A Course in Number Theory and Cryptography*, 2d ed. Berlin: Springer-Verlag, 1994.

[B99] Koblitz, N., "Elliptic curve cryptosystems," *Mathematics of Computation,* vol. 48, pp. 203–209, 1987.

[B100] Kocher, P., "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," N. Koblitz, Ed., *Advances in Cryptology—CRYPTO '96*, *Lecture Notes in Computer Science,* vol. 1109. Berlin: Springer-Verlag, 1996, pp. 104–113.

[B101] Kravitz, D., "Digital Signature Algorithm," U.S. Patent no. 5,231,668, July 1993.

[B102] Law, L. et al., "An Efficient Protocol for Authenticated Key Agreement," Technical Report CORR 98-05, Dept. of C&O, University of Waterloo, Canada, March 1998 (revised August 28, 1998).[27]

[B103] Lay, G., and H. Zimmer, "Constructing elliptic curves with given group order over large finite fields," *Algorithmic Number Theory: First International Symp., Lecture Notes in Computer Science,* vol. 877. Berlin: Springer-Verlag, 1994, pp. 250–263.

[B104] Lehmer, D., "Computer technology applied to the theory of numbers," W. LeVeque, Ed., *Studies in Number Theory*. Washington, D.C.: Mathematical Association of America, 1969.

[B105] Lenstra, H., Jr., "Factoring integers with elliptic curves," *Annals of Mathematics,* vol. 126, pp. 649–673, 1987.

[23] ITU-T publications are available from the International Telecommunications Union (http://www.itu.int/).
[24] Available at ftp://ftp.rsa.com/pub/cryptobytes/crypto2n1.pdf.
[25] Corrected version available at http://www.dice.ucl.ac.be/crypto/publications.html.
[26] Available from http://grouper.ieee.org/groups/1363/contrib.html.
[27] Available from http://www.cacr.math.uwaterloo.ca/.

[B106] Lim, C., and P. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup," B. Kaliski, Jr., Ed, *Advances in Cryptology—CRYPTO '97*, *Lecture Notes in Computer Science,* vol. 1294. Berlin: Springer-Verlag, 1997, pp. 249–263.

[B107] Liskov, M., and R. Silverman, "A statistical limited-knowledge proof for secure RSA keys," submitted to *Journal of Cryptology*, 1998.

[B108] MasterCard International, Inc. and Visa International Service Association, *SET Secure Electronic Transaction Specification*, May 31, 1997.[28]

[B109] Matsumoto, T. et al., "On seeking smart public-key-distribution systems," *The Transactions of the IECE of Japan,* vol. E69, pp. 99–106, 1986.

[B110] Maurer, U., "A universal statistical test for random bit generators," A. Menezes and S. Vanstone, Eds., *Advances in Cryptology—CRYPTO '90*, *Lecture Notes in Computer Science,* vol. 537. Springer-Verlag, 1991, pp. 409–420.

[B111] Maurer, U., "Fast generation of prime numbers and secure public-key cryptographic parameters," *Journal of Cryptology,* vol. 8, pp. 123–155, 1995.

[B112] Menezes, A., Ed., *Applications of Finite Fields*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1993.

[B113] Menezes, A., "Elliptic curve cryptosystems," *CryptoBytes,* vol. 1, no. 2, Summer 1995, RSA Laboratories.[29]

[B114] Menezes, A., *Elliptic Curve Public Key Cryptosystems*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1993.

[B115] Menezes, A. et al., *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1996.

[B116] Menezes, A. et al., "Reducing elliptic curve logarithms to logarithms in a finite field," *IEEE Transactions on Information Theory,* vol. 39, pp. 1639–1646, 1993.

[B117] Menezes, A. et al., "Some new key agreement protocols providing implicit authentication," in *Workshop Rec., 2nd Workshop on Selected Areas in Cryptography (SAC'95),* Ottawa, Canada, May 18-19, 1995, pp. 22-32.

[B118] Micali, S., and C. Schnorr, "Efficient, perfect polynomial random number generators," *Journal of Cryptology,* vol. 3, pp. 157–172, 1991.

[B119] Micali, S. et al., "The notion of security for probabilistic cryptosystems," *SIAM Journal on Computing,* vol. 17, pp. 412–426, 1988.

[B120] Mihailescu, P., "Fast generation of provable primes using search in arithmetic progressions," Y. Desmedt, Ed., *Advances in Cryptology—CRYPTO '94, Lecture Notes in Computer Science,* vol. 839. Berlin: Springer-Verlag, 1994, pp. 282–293.

[B121] Miller, V., "Use of elliptic curves in cryptography," H. Williams, Ed., *Advances in Cryptology—Crypto '85, Lecture Notes in Computer Science,* vol. 218. Berlin: Springer-Verlag, 1986, pp. 417–426.

[B122] Miyaji, A. et al., "New explicit conditions of elliptic curve traces for FR-reduction," *IEICE Transactions on Fundamentals*, vol. E84-A, pp. 1234–1243, 2001.

[B123] Morain, F., "Building cyclic elliptic curves modulo large primes," D. Davies, Ed., *Advances in Cryptology - EUROCRYPT '91*, *Lecture Notes in Computer Science,* vol. 547. Springer-Verlag, 1991, pp. 328-336.

[B124] National Institute of Standards and Technology, "Recommended elliptic curves for federal government use," draft 1999.[30]

[28] Available from http://www.setco.org/.
[29] Available at ftp://ftp.rsa.com/pub/cryptobytes/crypto1n2.pdf.
[30] Available from http://csrc.nist.gov/encryption/.

[B125] National Institute of Standards and Technology, *Special Publication 800-57: Recommendation for Key Management—Part 1: General*, revised 2011.

[B126] NIST Recommendation for Key Management, Part 1: General, NIST Special Publication 800-57, Aug. 2005.[31,32]

[B127] Nyberg, K., and R. Rueppel, "A new signature scheme based on the DSA giving message recovery," *1st ACM Conf. on Computer and Communcations Security*. New York: ACM Press, 1993, pp. 58–61.

[B128] Odlyzko, A., "The future of integer factorization," *CryptoBytes,* vol. 1, no. 2, summer 1995. RSA Laboratories.[33]

[B129] Oorschot, P. van, and M. Wiener, "Parallel collision search with applications to hash functions and discrete logarithms," *2nd ACM Conf. on Computer and Communications Security*. Washington, D.C.: ACM Press, 1994, pp. 210–218.

[B130] Pollard, J., "A Monte Carlo method for factorization," *BIT,* vol. 15, pp. 331–334, 1975.

[B131] Pollard, J., "Monte Carlo methods for index computation (mod *p*)," *Mathematics of Computation,* vol. 32, pp. 918–924, 1978.

[B132] Pollard, J., "Theorems on factorization and primality testing," *Proc. of the Cambridge Philosphical Society,* vol. 76, pp. 521-528, 1974.

[B133] Public Key Cryptography Standards (PKCS), PKCS #1 v1.5: RSA Encryption Standard, 1993.[34]

[B134] Public Key Cryptography Standards (PKCS), PKCS #1 v2.0: RSA Cryptography Standard, 1998.

[B135] Rabin, M., "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Institute of Technology Laboratory for Computer Science Technical Report 212 (MIT/LCS/TR-212), 1979.

[B136] RFC 1750, Randomness Recommendations for Security, Dec. 1994. [35]

[B137] RFC 2311, S/MIME Version 2 Message Specification, March 1998.

[B138] RFC 2312, S/MIME Version 2 Certificate Handling, March 1998.

[B139] Rivest, R. et al., "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM,* vol. 21, pp. 120–126, 1978.

[B140] Sakai, R., and M. Kasahara, "ID based cryptosystems with pairing on elliptic curve," Cryptology ePrint Archive, Report 2003/054, 2003.[36]

[B141] Satoh, T., and K. Araki, "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves," *Commentarii Mathematici Universitatis Sancti Pauli,* vol. 47, pp. 81-92, 1998. Errata: ibid. vol. 48, pp. 211–213, 1999.

[B142] Schirokauer, O., "Discrete logarithms and local units," *Philosophical Transactions of the Royal Society of London A*, vol. 345, pp. 409–423, 1993.

[B143] Schneier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2d ed. New York: John Wiley and Sons, 1995.

[B144] Schroeppel, R., and D. Eastlake, *Elliptic Curve Keys and Signatures in the DNS*, Oct. 2005, Internet draft.[37]

[31] Available at http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf.
[32] NIST publications are available from the National Institute of Standards and Technology (http://www.nist.gov/).
[33] Available at ftp://ftp.rsa.com/pub/cryptobytes/crypto1n2.pdf.
[34] Available from http://www.rsa.com/rsalabs/pubs/PKCS/.
[35] RFC publications are available from Requests for Comments (http://www.rfc-editor.org/). See also http://www.ietf.org/html.charters/smime-charter.html and http://www.ietf.org/ids.by.wg/smime.html for latest developments and drafts.
[36] Available at http://eprint.iacr.org/2003/054.
[37] Available at http://tools.ietf.org/html/draft-ietf-dnsext-ecc-key-08.

[B145] Schroeppel, R. et al., "Fast key exchange with elliptic curve systems," *Univ. of Arizona Comp. Sci. Tech. Report 95-03* (1995). A version also appears in D. Coppersmith, ed., *Advances in Cryptology—CRYPTO '95, Lecture Notes in Computer Science,* vol. 963. Berlin: Springer-Verlag, 1995, pp. 43–56.

[B146] Seroussi, G., "Compact representation of elliptic curve points over F_2^n," Research Manuscript. Palo Alto, CA: Hewlett-Packard Laboratories, April 1998.

[B147] Shamir, A., "RSA for paranoids," *CryptoBytes,* vol. 1, no. 3, 1995. RSA Laboratories.[38]

[B148] Shawe-Taylor, J., "Generating strong primes," *Electronics Letters,* vol. 22, pp. 875–877, 1986.

[B149] Silverman, J., *The Arithmetic of Elliptic Curves*. Berlin: Springer-Verlag, 1986.

[B150] Silverman, R., "The multiple polynomial quadratic sieve," *Mathematics of Computation,* vol. 48, 1987, pp. 329–339.

[B151] Smart, N., "Elliptic curve cryptosystems over small fields of odd characteristic," *Journal of Cryptology*, vol. 12, pp. 141–151, 1999.

[B152] Solo, D. et al., "(Extensible Markup Language) XML-Signature Syntax and Processing," March 2002, RFC 3275.[39]

[B153] Solo, D. et al., "Internet X.509 Certificate Request Message Format," Internet Engineering Task Force (IETF), PKIX working group, work in progress.[40]

[B154] Solo, D. et al., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," Internet Engineering Task Force (IETF), PKIX working group, work in progress.[41]

[B155] Stallings, W., *Cryptography and Network Security: Principles and Practice*, 2d ed. Upper Saddle River, NJ: Prentice-Hall, 1998.

[B156] Standards for Efficient Cryptography Group (SECG), *Recommended Elliptic Curve Domain Parameters*, GEC1, Sept. 1999.[42]

[B157] Stinson, D., *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.

[B158] Vaudenay, S., "Hidden collisions on DSS," N. Koblitz, Ed., *Advances in Cryptology—CRYPTO '96*, *Lecture Notes in Computer Science,* vol. 1109. Berlin: Springer-Verlag, 1996, pp. 83–88.

[B159] Wiener, M., "Cryptanalysis of short RSA secret exponents," *IEEE Transactions on Information Theory,* vol. 36, pp. 553–558, 1990.

[B160] Wiener, M., and R. Zuccherato, "Faster attacks on elliptic curve cryptosystems," S. Tavares and H. Meijer, Eds., *Selected Areas in Cryptography—SAC '98, Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1998.

[B161] Williams, H., "A modification on the RSA public-key encryption procedure," *IEEE Transactions of Information Theory,* vol. 26, pp. 726–729, 1980.

[B162] Williams, H., "A $p + 1$ method of factoring," *Mathematics of Computation,* vol. 39, pp. 225–234, 1982.

[B163] Yao, A., "Theory and applications of trapdoor functions," *Proceedings of the IEEE 23rd Annual Symposium on Foundations of Computer Science (FOCS '92)*, 1992, pp. 80–91.

[38] Available at ftp://ftp.rsa.com/pub/cryptobytes/crypto1n3.pdf.
[39] Available at http://www.ietf.org/rfc/rfc3275.txt.
[40] Available at http://www.ietf.org/ids.by.wg/pkix.html.
[41] Available at http://www.ietf.org/ids.by.wg/pkix.html.
[42] Available from http://www.secg.org/drafts.htm.