IEEE P1363.1™/D10 Draft Standard for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices

Prepared by the 1363 Working Group of the

C/MSC Committee

Abstract: Specifications of common public-key cryptographic techniques based on hard problems over lattices supplemental to those considered in IEEE 1363 and IEEE P1363a, including mathematical primitives for secret value (key) derivation, public-key encryption, identification and digital signatures, and cryptographic schemes based on those primitives. Specifications of related cryptographic parameters, public keys and private keys. Class of computer and communications systems is not restricted.

Keywords: Public key cryptography, encryption

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

PDF: ISBN 978-0-XXXX-XXXX-X STDXXXX
Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXX

This page is left blank intentionally.

Introduction

This introduction is not part of IEEE P1363.1/D10, Draft Standard for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices.

The P1363 project started as the "Standard for Rivest-Shamir-Adleman, Diffie-Hellman, and Related Public-Key Cryptography" with its first meeting in January 1994, following a strategic initiative by the Microprocessor Standards Committee to develop standards for cryptography. Over the next eight years, the working group produced a broad standard reflecting the state of the art in public key cryptography, including techniques from three major families of hard problems. In addition, the working group drafted an addendum that provides additional techniques from those three major families. A more thorough history of the P1363 working group and its contributions beyond the IEEE Std 1363-2000 are given in the Introduction to IEEE Std 1363-2000.

At the same time, new cryptographic research was producing additional families of cryptographic techniques. One of these families was the family of techniques based on hard problems over lattices. These techniques enjoy operating characteristics that make them attractive in certain implementation environments, and thus they have received considerable scrutiny and deployment.

As a result, the working group proposed a new project to standardize techniques from this family. This project was approved by the Microprocessor Standards Committee, and this current draft is the result of this project.

The following people have contributed to this draft standard:

Mark Etzel	Daniel Lieman, Editor (2001)	Nick Howgrave-Graham
Joseph H. Silverman	Ari Singer	William Whyte, Editor (2001-)

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard 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

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

Updating of IEEE documents

Users of IEEE standards 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. 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 Standards Association web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

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

Errata

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

Interpretations

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

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE 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 draft standard was completed, the 1363 Working Group had the following membership:

William Whyte, *Chair*

Don Johnson, *Vice Chair*

Matt Ball	Xavier Boyen	Mike Brenner

Daniel Brown
Mark Chimley
Andy Dancer
David Jablon
Don Johnson
Satoru Kanno
David Kravitz
Michael Markowitz
Luther Martin
Jim Randall
Roger Schlafly
Ari Singer
Terence Spies
Yongge Wang
William Whyte

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

(to be supplied by IEEE)

CONTENTS

Draft Standard for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices

1. Overview

1.1 Scope

Specifications of common public-key cryptographic techniques based on hard problems over lattices supplemental to those considered in IEEE 1363 and IEEE P1363a, including mathematical primitives for secret value (key) derivation, public-key encryption, identification and digital signatures, and cryptographic schemes based on those primitives. Specifications of related cryptographic parameters, public keys and private keys. Class of computer and communications systems is not restricted.

1.2 Purpose

The transition from paper to electronic media brings with it the need for electronic privacy and authenticity. Public-key cryptography offers fundamental technology addressing this need. Many alternative public-key techniques have been proposed, each with its own benefits. The IEEE 1363 Standard and P1363a project 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 project will specify cryptographic techniques based on hard problems over lattices. These techniques may offer tradeoffs in operating characteristics when compared with the methods already specified in IEEE 1363-2000 and draft P1363a. It is also intended that this project provide a second-generation framework for the description of cryptographic techniques, as compared to the initial framework provided in 1363-2000 and draft P1363a.

It is not the purpose of this project to mandate any particular set of public-key techniques or security requirements (including key sizes) for this or any family. Rather, the purpose is to provide: (1) a reference for specification of a variety of techniques from which applications may select, (2) the relevant number-theoretic background, and (3) extensive discussion of security and implementation considerations so that a solution provider can choose appropriate security requirements for itself.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

FIPS 180-2, *Secure Hash Standard*, Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, Virginia, August 26, 2002 (supersedes FIPS PUB 180-1). Available at `http://csrc.nist.gov/CryptoToolkit/Hash.html`.
ISO/IEC 10118-3:1998 Information Technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.

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

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

NOTE 3—Bibliography is provided in Annex B.

3. Definitions

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

3.1 **Algorithm:** A clearly specified mathematical process for computation; a set of rules which, if followed, will give a prescribed result.

3.2 **Asymmetric Cryptographic Algorithm:** A cryptographic algorithm that uses two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key.

3.3 **Authentication (of a message):** The act of determining that a message has not been changed since leaving its point of origin. The identity of the originator is implicitly verified.

3.4 **Authentication of Ownership:** The assurance that a given, identified party intends to be associated with a given public key. May also include assurance that the party possesses the corresponding private key (see IEEE Std 1363-2000, Annex D.3.2, for more information).

3.5 **Big Modulus:** The big modulus q is used to define the larger polynomial ring. The modulus q can generally be taken to be any value that is relatively prime in the ring to the small modulus p.

3.6 **Birthday Paradox:** For a category size of 365 (the days in a year), after only 23 people are gathered, the probability is greater than 0.5 that at least two people have a common birthday (month and day). The reason is that among 23 people, there are 23*(23-1)/2 = 253 pairs of people, each with a 1/365 chance of having matching birthdays. The chance of no matching birthday is therefore (364/365)253 ~ 0.4995. In general, any case where the criterion for success is to find a collision (two matching values) rather than a hit (one value which matches a pre-selected one) will display this pairing property, so that the size of the space that must be searched for success is about the square root of the size of the space of all possible values.

3.7 **Bit Length:** See: length.

3.8 **Bit String:** An ordered sequence of 0's and 1's. The left-most bit is the most-significant bit of the string. The right-most bit is the least-significant bit of the string. A bit and a bit string of length 1 are equivalent for all purposes of this standard.

3.9 **Blinding Polynomial:** In this standard, the ciphertext e is generated according to the equation $e = r*h + m'$, where h is the public key, m' is the message representative, and r is a pseudorandomly generated “blinding polynomial”

3.10 **Blinding Polynomial Generation Methods:** In the encryption schemes in this document, a blinding polynomial generation method (LBP-BPGM) is used to generate a blinding polynomial r from the padded message pm in order to provide plaintext awareness.

3.11 **Blinding Polynomial Space:** The space that a LBP-BPGM selects from. Usually defined implicitly by the definition of the LBP-BPGM.

3.12 **Certificate:** The public key and identity of an entity together with some other information rendered unforgeable by signing the certificate with the private key of the certifying authority, which issued that certificate.

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

3.14 **Composite:** An integer which has at least two prime factors.

3.15 **Confidentiality:** The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

3.16 **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

3.17 **Cryptographic Family:** A set of cryptographic techniques in similar mathematical settings. For example, this standard presents a single family of techniques based on the underlying hard problems of finding a short vector and a close vector in a lattice.

3.18 **Cryptographic Hash Function:** See hash function.

3.19 **Cryptographic Key (Key):** A parameter that determines the operation of a cryptographic function such as: the transformation from plain text to cipher text and vice versa; synchronized generation of keying material; digital signature computation or validation.

3.20 **Cryptography:** The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification, prevent its unauthorized use or a combination thereof.

3.21 **Data Integrity:** A property whereby data has not been altered or destroyed.

3.22 **Decrypt:** To produce plaintext (readable) from ciphertext (unreadable). Contrast: encrypt. See also: ciphertext; encryption; plaintext.

3.23 **Dimension:** The dimension N identifies the dimension of the convolution polynomial ring used. The dimension of the associated lattice problem is 2N. Elements of the ring are represented as polynomials of degree N – 1.

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

3.25 **Domain Parameter Validation:** the process of ensuring or verifying that a set of domain parameters is valid. See also: domain parameters; key validation; valid domain parameters.

3.26 **Encrypt:** to produce ciphertext (unreadable) from plaintext (readable). Contrast: decrypt. See also: ciphertext; encryption; plaintext.

3.27 **Encryption Primitives:** The encryption primitive is the fundamental building block for the encryption operation. In public key cryptography, an encryption primitive scrambles data using a public key such that only the holder of the private key can directly perform the unscrambling operation; in other words, it provides security against ciphertext-only attacks by passive attackers.

3.28 **Encryption Scheme:** A means for providing encryption, based on an encryption primitive, that is secure against both active and passive attackers. A secure encryption scheme will typically provide semantic security (an attacker who knows that one of two messages has been encrypted will find it computationally infeasible to determine which) against an attacker who can make polynomially many queries to a decryption oracle.

3.29 **Entity:** A participant in any of the schemes in this standard. The words “entity” and “party” are used interchangeably. This definition may admit many interpretations: it may or may not be limited to the necessary computational elements; it may or may not include or act on behalf of a legal entity. The particular interpretation chosen will not affect operation of the key agreement schemes.

3.30 **Exclusive OR:** A mathematical bit-wise operation, symbol $\oplus$, defined as:

$$0 \oplus 0 = 0,$$
$$0 \oplus 1 = 1,$$
$$1 \oplus 0 = 1, \text{ and}$$
$$1 \oplus 1 = 0.$$

Equivalent to binary addition without carry. May also be applied to bit strings: the XOR of two bit strings of equal length is the concatenation of the XORs of the corresponding elements of the bit strings.

3.31 **Family:** See: cryptographic family.

3.32 **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).

3.33 **Finite Field:** a field in which there are only a finite number of quantities.

3.34 **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.

3.35 **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.

3.36 **Hash Function:** A function which maps a bit string of arbitrary length to a fixed-length bit string and satisfies the following properties:
It is computationally infeasible to find any input which maps to any pre-specified output;
It is computationally infeasible to find any two distinct inputs which map to the same output.

3.37 **Hash Value**: The result of applying a hash function to a message.

3.38 **Index Generation Function**: An IGF is a function that is seeded once, can be called multiple times, and produces statistically independent integers on each call.

3.39 **Key:** See cryptographic key.

3.40 **Key Confirmation:** The assurance of the legitimate participants in a key establishment protocol that the intended recipients of the shared key actually posses the shared key.

3.41 **Key Derivation:** The process of deriving one or more session keys from a shared secret and (possibly) other, public information. Such a function can be constructed from a one-way hash function such as SHA-1.

3.42 **Key Encrypting Key (KK):** A key used exclusively to encrypt and decrypt keys.

3.43 **Key Establishment:** A protocol that reveals a secret key to its legitimate participants for cryptographic use.

3.44 **Key Generation Primitive:** A method used to generate a key pair.

3.45 **Key Management:** The generation, storage, secure distribution and application of keying material in accordance with a security policy.

3.46 **Key Pair:** When used in public key cryptography, a private key and its corresponding public key. 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.

3.47 **Key Transport:** A key establishment protocol under which the secret key is determined by the initiating party.

3.48 **Key Validation:** the process of ensuring or verifying that a key conforms to the arithmetic requirements for such a key in order to thwart certain types of attacks. See also: domain parameter validation; public/private key pair; valid key; valid key pair.

3.49 **Keying Material:** The data (e.g., keys, certificates and initialization vectors) necessary to establish and maintain cryptographic keying relationships.

3.50 **Known-Key Security:** Known-key security for Party U implies that the key agreed upon will not be compromised by the compromise of the other session keys. If each ephemeral key is used only to compute a single session key, then known-key security may be achieved.

3.51 **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.

3.52 **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.

3.53 **Lattice Based Polynomial Public Key Encryption**: The encryption mechanisms described in this standard.

3.54 **Least Significant:** See: last bit; last octet.

3.55 **Leftmost Bit:** See: first bit.

3.56 **Leftmost Octet:** See: first octet.

3.57 **Length:** (1) Length of a bit string is the number of bits in the string. (2) Length of an octet string is the number of octets in the string. (3) Length in bits of a nonnegative integer n is $\lfloor \log2 (n + 1) \rfloor$ (i.e., the number of bits in the integer's binary representation). (4) Length in octets of a nonnegative integer n is $\lfloor \log256 (n + 1) \rfloor$ (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, and its length in octets is 2.

3.58 **Mask Generation Function:** An MGF is a construction built around a hash function that produces an arbitrary-length output string, possibly longer than the output of the underlying hash function.

3.59 **Message Authentication Code (MAC):** A cryptographic value which is the results of passing a financial message through the message authentication algorithm using a specific key.

3.60 **Message Length Encoding Length:** In SVES, the length of the message that is to be encrypted is encoded in the padded message. The length of the field that represents the length of the message, called the message length encoding length, is represented by the parameter lLen. For all parameter sets in this document lLen is set to 1.

3.61 **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 and uniquely linked to that message. See also: encryption scheme; digital signature scheme.

3.62 **Modular Lattice:** A lattice in which (among other things) all values are integers reduced mod *q*.

3.63 **Most Significant:** See: first bit; first octet.

3.64 **Norm:** A measure of the "size" of a vector or polynomial.

3.65 **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. Also commonly known as a byte. See also: bit string.

3.66 **Octet String:** An ordered sequence of octets. See also: octet.

3.67 **Owner:** The entity whose identity is associated with a key pair.

3.68 **Parameters:** See: domain parameters.

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

3.70 **Polynomial Index Generation Constant:** A value used when generating a random number in the range [0, N-1], to eliminate bias without impacting efficiency.

3.71 **Prime Number:** An integer that is greater than 1 and divisible only by 1 and itself.

3.72 **Primitives:** Cryptographic primitives used in the SVES encryption scheme include key generation primitives, encryption primitives and decryption primitives.

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

3.74 **Private Key Space:** The space from which a key generation primitive selects the private key.

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

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

3.77 **Public Key Space:** The space from which a key generation primitive selects the public key.

3.78 **Public Key Validation:** See key validation.

3.79 **Public/Private Key Pair:** See key pair.

3.80 **Salt Size:** In this standard, the salt size db is the number of random bits that shall be used to pad the message during encryption, to provide for semantic security.

3.81 **Rightmost Bit:** See: last bit.

3.82 **Rightmost Octet:** See: last octet.

3.83 **Ring:** a setting in which addition, subtraction, and multiplication are possible, and division by a given nonzero quantity may or may not be possible. A field is a special case of a ring. See also: field.

3.84 **Ring Element:** in general, an element in a ring. In the context of this standard, a *binary N-ring element* refers to an element in the ring $(\mathbf{Z}/2\mathbf{Z})[X]/(X^N - 1)$, which is to say a binary polynomial of degree N-1 or an array of N binary elements. A *(q, N)-ring element* refers to an element in the ring $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$, which is to say a polynomial of degree N-1 with coefficients reduced mod q or an array of N elements each taken mod q.

3.85 **Scheme Options:** Scheme options consist of parameters and algorithms that do not affect the key space (i.e. that are not domain parameters), but that must be agreed upon in order to implement the encryption scheme.

3.86 **Secret Key:** a key used in symmetric cryptography; needs to be known to all legitimate participating parties involved, but cannot be known to an adversary. Contrast: public/private key pair. See also: key agreement; shared secret key; symmetric cryptography.

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

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

3.89 **Shared Secret Value:** a secret value shared by two parties, usually during a key agreement scheme. See also: key agreement; secret value.

3.90 **Signature:** See: digital signature.

3.91 **Small Modulus:** In LBP-PKE, the small modulus p is used for key generation and for modular reduction during decryption.

3.92 **Statistically Unique:** For the generation of n-bit quantities, the probability of two values repeating is less than or equal to the probability of two n-bit random quantities repeating. More formally, an element chosen from a finite set S of n elements is said to be "statistically unique" if the process that governs the selection of this element provides a guarantee that, for any integer $L \leq n$, the probability that all of the first L selected elements are different is no smaller than the probability of this happening when the elements are drawn uniformly randomly from S. The latter probability is equal L)!nL ˜.to n!(n

3.93 **SVES:** Short Vector Encryption Scheme – the encryption scheme defined in this document.

3.94 **Symmetric Cryptographic Algorithm:** A cryptographic algorithm that uses one shared key, a secret key. The key must be kept secret between the two communicating parties. The same key is used for both encryption and decryption.

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

3.96 **Symmetric Key:** A cryptographic key that is used in symmetric cryptographic algorithms. The same symmetric key that is used for encryption is also used for decryption.

3.97 **User:** A party that uses a public key.

3.98 **Valid Domain Parameters:** a set of domain parameters that satisfies the specific mathematical definition for the set of domain parameters of its family. While a set of mathematical objects may have the general structure of a set of domain parameters, it may not actually satisfy the definition (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.

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

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

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

3.102 **Verify:** In relation to a Digital Signature means to determine accurately: (1) that the Digital Signature was created during the operational period of a valid Certificate by the private key corresponding to the public-key listed in the Certificate; and (2) the message has not been altered since its Digital Signature was created.

4. Types of cryptographic techniques

4.1 General model

As stated in Clause 1, the purpose of this standard is to provide a reference for specifications of a variety of common public-key cryptographic techniques from which applications may select. Different types of cryptographic techniques can be viewed abstractly according to the following three-level general model.

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

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

This standard contains only specifications of schemes.

4.2 Schemes

The following types of schemes are defined in this standard:

- Encryption Schemes (ES), in which any party can encrypt a message using a recipient’s public key, and only the recipient can decrypt the message by using its corresponding private key. Encryption schemes may be used for establishing secret keys to be used in symmetric cryptography.

Schemes in this standard are presented in a general form based on certain primitives and additional methods. For example, the encryption scheme defined in this standard is based on a key generation primitive, a decryption primitive, and a blinding polynomial generation 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 performed properly, and in some cases validation also needs to be performed. While outside the scope of this standard, proper key management is essential for security.

An *Encryption Scheme* is specified by providing the following:

- Name
- Type (e.g. Asymmetric Public-key Encryption Scheme)
- Options (Key Type, Primitives, Parameters)
- Operations

  - Key Pair Generation
  - Key Pair Validation
  - Public Key Validation
  - Encryption Operation
    - Input
    - Output
  - Decryption Operation
    - Input
    - Output

An encryption scheme specification may also include the following:

- Security Considerations
- Implementation Considerations
- Related Standards

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.

4.3 Additional methods

This standard specifies the following additional methods:

- Blinding Polynomial Generation Methods, which are components of encryption schemes.
- Auxiliary Functions, which are building blocks for other additional methods.
  - Index generation functions
  - Mask Generation Functions
  - Hash Functions, which are used as the core of Index generation functions and of Mask Generation Functions.

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

4.4 Algorithm specification conventions

When specifying an algorithm or method, this standard uses four parts to specify different aspects of the algorithm. They are as follows:

Components, such as choice of IGF, are parameters that are specified before the beginning of the operation and that are not specific to the particular algorithm call. Components tend to be kept fixed for multiple users and multiple instances of the algorithm call and need not be explicitly specified if they are implicitly known (e.g. if they are defined within a selected object identifier (OID)).

Inputs, such as keys and messages, are values that must be specified for each algorithm call.

Outputs, such as ciphertext, are the result of transformations on the inputs.

Operations specify the transformations that are performed on the data to arrive at the output. Throughout the standard, the operations are defined as a sequence of steps. A conformant implementation may perform the operations using any sequence of steps that always produces the same output as the sequence in this standard. Caution should be taken to ensure that intermediate values are not revealed, however, as they may compromise the security of the algorithms.

5. Mathematical conventions

5.1 Mathematical notation and abbreviated terms

When referring to mathematical objects and data objects in this standard, the following notation is used. Throughout the document, numbers at the end of variable names are used to distinguish different, but related values (e.g. *df1*, *df2*, *df3* or *Dmin1*, *Dmin2*, etc.).

0	Denotes the integer 0, the bit 0, or the additive identity (the element zero) of a ring
1	Denotes the integer 1, the bit 1, or the multiplicative identity (the element one) of a ring
*	Indicates the convolution product of two polynomials and is also used to indicate multiplication of integers
⊕ or XOR	Exclusive OR function
||	Concatenation. A||B is the concatenation of the octet strings A and B where the leading octet of A is the leading octet of A||B and the trailing octet of B is the trailing octet of A||B.
:=	Initialization. a := b means initialize or set the value of a equal to the value of b.
A	Lower-bound decryption coefficient, used in decryption process to reduce into correct interval
BRE2OSP	Binary Ring Element to Octet String Conversion Primitive
BS2IP	Bit String to Integer Conversion Primitive
BS2REP	Bit String to Ring Element Conversion Primitive
BS2ROSP	Bit String to Right-padded Octet String Conversion Primitive

BPGM	Blinding Polynomial Generation Method
ceil[.] or $\lceil . \rceil$	Ceiling function (i.e. the smallest integer greater than or equal to the contents of [.])
db	The number of random bits used as input for encryption
df	An integer specifying the number of ones in the polynomials that comprise the private key value *f* (also specified as *df*1, *df*2, and *df*3, or as *dF*)
dg	An integer specifying the number of ones in the polynomials that comprise the temporary polynomial *g* (often specified as *dG*)
DP	Decryption Primitive
dr	An integer specifying the number of ones in the blinding polynomial *r* in SVES. (also specified as *dr*1, *dr*2, and *dr*3)
e	Encrypted message representative, a polynomial, computed by an encryption primitive
E	Encrypted message, an octet string.
ES	(Asymmetric) encryption scheme.
f	Private key in SVES.
F	In SVES, a polynomial that is used to calculate the value *f* when *f*=1+*pF*.
floor[.] or $\lfloor . \rfloor$	Floor function (i.e. the largest integer less than or equal to the contents of [.])
g	In SVES, a temporary polynomial used in the key generation process.
GCD(a, b)	Greatest Common Divisor of two non-negative integers a and b.
h	Public key
Hash( )	A cryptographic hash function computed on the contents of ( )
hLen	Length in octets of a hash value.
i	An integer
I2BSP	Integer to Bit String Conversion Primitive
I2OSP	Integer to Octet String Conversion Primitive
IGF( )	An index generation function seeded with the contents of ( )
IGF-MGF1	An index generation function based on the MGF1 construction.
k	Security level in bits.
KGP	Key Generation Primitive

LBP-BPGM1	Blinding polynomial generation method for generating binary blinding polynomials
LBP-BPGM2	Blinding polynomial generation method for generating product-form blinding polynomials
LBP-DP1	Decryption primitive for use with lattice based polynomial public key decryption
LBP-KGP1	Random Key Generation Primitive
LBP-KGP2	Random Low Hamming Weight Key Generation Primitive
LBP-PKE	Lattice-Based Polynomial Public Key Encryption
m	The message, an octet string, which is encrypted in SVES.
M	In SVES, the padded and formatted message representative octet string used during encryption and decryption.
m'	The message representative polynomial which is submitted to the encryption primitive in the SVES encryption scheme.
MAC	Message authentication code.
MGF( )	A mask generation function seeded with the contents of ( )
MGF1	A mask generation function based on hashing a seed concatenated with a counter.
mod *q*	Used to reduce the coefficients of a polynomial into some interval of length *q*
mod *p*	Used to reduce a polynomial to an element of the polynomial ring mod p
MPM	Message Padding Method
MRGM	Message Representative Generation Method
N	Dimension of the polynomial ring used (i.e. polynomials are up to degree *N*-1)
OS2BREP	Octet String to Binary Ring Element Conversion Primitive
OS2IP	Octet String to Integer Conversion Primitive
OS2REP	Octet String to Ring Element Conversion Primitive
p	"Small" modulus, an integer or a polynomial
q	"Big" modulus, usually an integer
r	In LBP-PKE, the encryption blinding polynomial (generated from the hash of the padded message M in SVES)
RE2BSP	Ring Element to Bit String Conversion Primitive
RE2OSP	Ring Element to Octet String Conversion Primitive

ROS2BSP	Right-padded Octet String to Bit String Conversion Primitive
SVDP	Short Vector Decryption Primitive
SVES	Short Vector Encryption Scheme
x	The integer input to or output from integer conversion primitives
X	The indeterminate used in polynomials
$\boldsymbol{Z}$	The ring of integers
$\boldsymbol{Z}_q$	The ring of integers mod q.

6. Polynomial representation and operations

6.1 Introduction

The cryptographic techniques specified in this standard require arithmetic in quotient polynomial rings, also called convolution polynomial rings. Intuitively, these algebraic objects consist of polynomials with integer coefficients. Manipulation of these ring elements is accomplished by polynomial arithmetic modulo a fixed polynomial: $X^N - 1$ in this standard.

6.2 Polynomial representation

Typically in mathematical literature, a polynomial a in X is denoted $a(X)$. In this standard, when the meaning is clear from the context, polynomials a in the variable X will simply be denoted a. Further, all polynomials used in this standard have degree $N - 1$, unless otherwise noted. In addition, given a polynomial a, a variable denoted a_i, where i is an integer, represents the coefficient of a of degree i. In other words, the polynomial denoted a represents the polynomial $a(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots + a_iX^i + \ldots + a_{N-1}X^{N-1}$, unless otherwise specified.

6.3 Polynomial operations

6.3.1 Polynomial multiplication

Let **Z** be the ring of integers. The polynomial ring over **Z**, denoted $\mathbf{Z}[X]$, is the set of all polynomials with coefficients in the integers. The *convolution polynomial ring (over* **Z***) of degree N* is the quotient ring $\mathbf{Z}[X]/(X^N - 1)$. The product c of two polynomials $a,b \in \mathbf{Z}[X]/(X^N - 1)$ is given by the formula

$$c(X) = a(X) * b(X) \quad \text{with} \quad c_k = \sum_{i+j \equiv k \pmod N} a_i b_j \ .$$

All multiplications of polynomials a and b, represented as $a*b$, are taken to occur in the ring $\mathbf{Z}[X]/(X^N - 1)$ unless otherwise noted.

6.3.2 Reduction of a Polynomial mod q

Throughout the document, polynomials are taken *mod q*, where q is an integer. To reduce a polynomial *mod q*, one simply reduces each of the coefficients independently *mod q* into the appropriate (specified) interval.

6.3.3 Inversion in $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$

For certain cryptographic operations such as key generation, it is necessary to take the inverse of a polynomial in $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$. This clause describes the algorithms necessary for inversion in this ring.

6.3.3.1 The Polynomial Division Algorithm in $\mathbf{Z}_p[X]$

This algorithm divides one polynomial by another polynomial in the ring of polynomials with integer coefficients modulo a prime p. All convolution operations occur in the ring $\mathbf{Z}_p[X]$ in this algorithm (i.e. there is no modular reduction of the powers of the polynomials).

Algorithm 1 – Polynomial Division Algorithm in $\mathrm{Z_p[X]}$

Input: A prime p, a polynomial a in $\mathbf{Z}_p[X]$ and a polynomial b in $\mathbf{Z}_p[X]$ of degree N-1 whose leading coefficient b_N is not 0.

Output: Polynomials q and r in $\mathbf{Z}_p[X]$ satisfying $a = b * q + r$ and deg r < deg b.

Operation: Polynomial Division Algorithm in $\mathrm{Z}_p[X]$ shall be computed by the following or an equivalent sequence of steps;

```
a)  Set $r := a$ and $q := 0$
b)  Set $u := b_N^{-1} \bmod p$
c)  While deg $r >= N$ do
    1)  Set $d :=$ deg $r(X)$
    2)  Set $v := u * r_d * X^{(d-N)}$
    3)  Set $r := r - v * b$
    4)  Set $q := q + v$
d)  Return $q, r$
```

6.3.3.2 The Extended Euclidean Algorithm in $\mathrm{Z_p[X]}$

The Extended Euclidean Algorithm finds a greatest common divisor d (there may be more than one that are constant multiples of each other) of two polynomials a and b in $\mathbf{Z}_p[X]$ and polynomials u and v such that $a*u + b*v = d$. All convolution operations occur in the ring $\mathbf{Z}_p[X]$ in this algorithm (i.e. there is no modular reduction of the powers of the polynomials).

Algorithm 2 – Extended Euclidean Algorithm in $\mathrm{Z_p[X]}$

Algorithm 2 – Extended Euclidean Algorithm in $Z_p[X]$

Input: A prime p and polynomials a and b in $\mathbf{Z}_p[X]$ with a and b not both zero.

Output: Polynomials u, v, d in $\mathbf{Z}_p[X]$ with $d = \mathrm{GCD}(a, b)$ and $a*u + b*v = d$.

Operation: Extended Euclidean Algorithm in $Z_p[X]$ shall be computed by the following or an equivalent sequence of steps;

```
a)  If $b = 0$ then return $(1,0,a)$
b)  Set $u := 1$
c)  Set $d := a$
d)  Set $v_1 := 0$
e)  Set $v_3 := b$
f)  While $v_3 \neq 0$ do
    1)  Use the division algorithm (6.3.3.1) to write $d = v_3*q + t_3$ with $\deg t_3 < \deg v_3$
    2)  Set $t_1 := u - q*v_1$
    3)  Set $u := v_1$
    4)  Set $d := v_3$
    5)  Set $v_1 := t_1$
    6)  Set $v_3 := t_3$
g)  Set $v := (d - a*u)/b$  [This division is exact, i.e., the remainder is 0]
h)  Return $(u,v,d)$
```

6.3.3.3 Inverses in $Z_p[X]/(X^N - 1)$

The Extended Euclidean Algorithm may be used to find the inverse of a polynomial a in $\mathbf{Z}_p[X]/(X^N - 1)$ if the inverse exists. The condition for the inverse to exist is that $\mathrm{GCD}(a, X^N - 1)$ should be a polynomial of degree 0 (i.e. a constant). All convolution operations occur in the ring $\mathbf{Z}_p[X]/(X^N - 1)$ in this algorithm.

Algorithm 3 – Inverses in $Z_p[X]/(X_N - 1)$

Input: A prime p, a positive integer N and a polynomial a in $\mathbf{Z}_p[X]/(X^N - 1)$.

Output: A polynomial b satisfying $a*b = 1$ in $\mathbf{Z}_p[X]/(X^N - 1)$ if a is invertible in $\mathbf{Z}_p[X]/(X^N - 1)$, otherwise FALSE.

Operation: Inverses in $Z_p[X]/(X^N - 1)$ shall be computed by the following or an equivalent sequence of steps;

```
a)  Run the Extended Euclidean Algorithm (6.3.3.2) with input $a$ and $(X^N - 1)$. Let $(u, v, d)$
    be the output, such that $a*u + (X^N - 1)*v = d = \mathrm{GCD}(a, (X^N - 1))$.
b)  If $\deg d = 0$
```

Algorithm 3 – Inverses in $Z_p[X]/(X_N – 1)$

```
c)  Return $b = d^{-1}$ (mod $p$) * $u$
d)  Else return FALSE
```

6.3.3.4 Inverses in $Z_{p^\wedge r}[X]/(X^N – 1)$

For key generation in this standard it is necessary to calculate inverses in $Z_a[X]/(X^N – 1)$, where q is a power of 2. In this case, the Inversion Algorithm (6.3.3.3) may be used to find the inverse of $a(X)$ in the quotient ring $(R/2R)[X]/(M(X))$. Then the following algorithm may be used to lift it to an inverse of $a(X)$ in the quotient ring $(R/p^eR)[X]/(M(X))$ with higher powers of the prime 2 (or any prime p).

Algorithm 4 – Inverses in Zp[X]/(XN – 1)

Input. A prime p in a Euclidean ring R, a monic polynomial $M(X)$ ε $R[X]$, a polynomial $a(X)$ ε $R[X]$, and an exponent e.

Output. An inverse $b(X)$ of $a(X)$ in the ring $(R/p^eR)[X]/(M(X))$ if the inverse exists, otherwise FALSE.

```
a)  Use the Inversion Algorithm 6.3.3.4 to compute a polynomial $b(X)$ ε $R[X]$ that gives an
    inverse of $a(X)$ in $(R/pR)[X]/(M(X))$. Return FALSE if the inverse does not exist. [The
    Inversion Algorithm may be applied here because $R/pR$ is a field, and so $(R/pR)[X]$ is a
    Euclidean ring.]
b)  Set $n \leftarrow 2$
c)  While $e > 0$ do
d)  $b(X) \leftarrow 2*b(X) – a(X)*b(X)^2$   (mod $M(X)$), with coefficients computed modulo $p^n$
e)  Set e $\leftarrow \lfloor e/2 \rfloor$
f)  Set $n \leftarrow 2*n$
g)  Return $b(X)$ mod $M(X)$ with coefficients computed modulo $p^e$.
```

7. Data Types and Conversions

7.1 Bit Strings and Octet Strings

As usual, a **bit** is defined to be an element of the set {0, 1}. A **bit string** is defined to be an ordered array of bits. A **byte** (also called an **octet**) is defined to be a bit string of length 8. A **byte string** (also called an **octet string**) is an ordered array of bytes. The terms **first** and **last**, **leftmost** and **rightmost**, **most significant** and **least significant**, and **leading** and **trailing** are used to distinguish the ends of these sequences (**first**, **leftmost**, **most significant** and **leading** are equivalent; **last**, **rightmost**, **least significant** and **trailing** are equivalent). Within a byte, we additionally refer to the **high-order** and **low-order** bits, where **high-order** is equivalent to **first** and **low-order** is equivalent to **last**.

Note that when a string is represented as a sequence, it may be indexed from left to right or from right to left, starting with any index. For example, consider the octet string of two octets: 2a 1b. This corresponds to the bit string 0010 1010 0001 1011. No matter what indexing system is used, the first octet is still 2a, the first bit is still 0, the last octet is still 1b, and the last bit is still 1. The high-order bit of the second octet is 0; the low-order bit of the second octet is 1.

When a bit string or a octet string is being encoded into a polynomial with coefficients reduced mod q (a "ring element"), where q is usually either 128 or 256, the integer coefficients are mapped individually to bit or octet strings, which are then concatenated. This mapping and its reverse are described in the conversion primitives OS2REP, BS2REP, RE2OSP and RE2BSP in 7.5 and 7.6.

This standard does not specify a single algorithm for converting from bit/octet strings to trinary polynomials in an unbiased and reversible fashion. Instead, the standard uses two algorithms, which are defined inline in the techniques that use them. One algorithm is reversible but biased; the other is unbiased but non-reversible.

7.2 Converting Between Integers and Bit Strings (I2BSP and BS2IP)

7.2.1 Integer to Bit String Primitive (I2BSP)

I2OSP converts a nonnegative integer to a bit string of a specified length.

Algorithm 5 – I2BSP

Input: *i*, nonnegative integer to be converted; *bLen*, intended length of the resulting bit string

Output: *B*, corresponding bit string of length *bLen*

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

a) If $x \geq 2^{xLen}$, output "integer too large" and stop.

b) Write the integer *x* in its unique *xLen*-bit representation in base 2:
$$x = x_{xLen-1} \cdot 2^{xLen-1} + x_{xLen-2} \cdot 2^{xLen-2} + \ldots + x_1 \cdot 2 + x_0$$
where $x_i = 0$ or 1 (note that one or more leading bits will be zero if *x* is less than 2^{xLen-1}).

c) Output the bit string$x_{xLen-1}\ x_{xLen-2}\ \ldots\ x_1\ x_0$.

7.2.2 Bit String to Integer Primitive (BS2IP)

BS2IP converts a bit string to a nonnegative integer.

Algorithm 6 – BS2IP

Input: *B*, bit string to be converted (*bLen* is used to denote the length of *B*)

Output: *x*, corresponding nonnegative integer

Algorithm 6 – BS2IP

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

a) If B is of length 0, output 0.

b) Let $b_{bLen-1}\ b_{bLen-2}\ \ldots\ b_1\ b_0$ be the bits of B from leftmost to rightmost.

c) Let $x = b_{bLen-1} \cdot 2^{\,bLen-1} + b_{bLen-2} \cdot 2^{\,bLen-2} + \ldots + b_1 \cdot 2 + b_0$.

d) Output x.

7.3 Converting Between Integers and Octet Strings (I2OSP and OS2IP)

7.3.1 Integer to Octet String Primitive (I2OSP)

I2OSP converts a nonnegative integer to an octet string of a specified length.

Algorithm 7 – I2OSP

Input: x, nonnegative integer to be converted; $oLen$, intended length of the resulting octet string

Output: O, corresponding octet string of length $oLen$

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

a) If $x \geq 256^{\,oLen}$, output "integer too large" and stop.

b) Write the integer x in its unique $oLen$-digit representation in base 256:
$$x = o_{oLen-1} \cdot 256^{\,oLen-1} + o_{oLen-2} \cdot 256^{\,oLen-2} + \ldots + o_1 \cdot 256 + o_0$$
where $0 \leq o_i < 256$ (note that one or more leading digits will be zero if o is less than 256^{oLen-1}).

c) For for $1 \leq x \leq oLen$, let the octet O_i be the concatenation of the bits in the integer representation of o_{oLen-i} , where left-most bit of the octet is the high order bit of the binary representation. Output the octet string
$$O = O_1\ O_2\ \ldots\ O_{oLen}\ .$$

NOTE—As an example, the integer 944 has the three-digit representation $944 = 0 \cdot 256^{\,2} + 3 \cdot 256 + 178$. The corresponding octet string, expressed in integer values, is 0 3 178; as binary values, it is

```
00000000  00000011  10110010
```

and in hexadecimal it is 00 03 b2.

7.3.2 Octet String to Integer Primitive (OS2IP)

OS2IP converts an octet string to a nonnegative integer.

Algorithm 8 – OS2IP

Input: x, nonnegative integer to be converted; $oLen$, intended length of the resulting octet string

Algorithm 8 – OS2IP

Output: *O*, corresponding octet string of length *oLen*

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

a) If *O* is of length 0, output 0.

b) Let $O_1\ O_2\ \dots\ O_{oLen}$ be the octets of *O* from first to last, and let o_{oLen-j} be the integer value of the octet O_j for 1 □ *j* □ *oLen*, where the integer value is represented as an octet (x.e., an eight-bit string) most significant bit first.

c) Output $x = o_{oLen-1} \cdot 256^{\,oLen-1} + o_{oLen-2} \cdot 256^{\,oLen-2} + \dots + o_1 \cdot 256 + o_0$.

7.4 Converting Between Bit Strings and Right-Padded Octet Strings (BS2ROSP and ROS2BSP)

This clause gives the primitives used to convert between bit strings and right-padded octet strings.

7.4.1 Bit String to Right-Padded Octet String Primitive (BS2ROSP)

Algorithm 9 – BS2ROSP

Input: *B*: bit string to be converted; *oLen*: intended length of the resulting octet string

Output: *O*, corresponding octet string of length *oLen*

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

a) Set *bLen* equal to the length of *x* in bits.

b) If *bLen* > 8**oLen*, output “input too long” and stop.

c) Append (8**oLen* – *bLen*) zero bits to the end of *x*.

d) Let $b_0\ b_1\ \dots\ b_{xLen-2}\ b_{xLen-1}$ be the bits of *B* from first to last. For $0 \le i < oLen - 1$, let the octet $O_i = b_{8i}\ b_{8i+1}\ \dots\ b_{8i+7}$. Output the octet string $O = O_0\ O_1\ \dots\ O_{oLen\text{-}1}$.

7.4.2 Right-Padded Octet String to Bit String Primitive (ROS2BSP)

ROS2BSP converts an octet string to a bit string of a specified length.

Algorithm 10 – ROS2BSP

Input: *O*: octet string to be converted; *bLen*: intended length of the resulting bit string

Output: *B*: corresponding bit string of length *bLen*

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

a) Set *oLen* equal to the length of *O* in octets.

b) If *bLen* > 8**oLen*, output “input too short” and stop.

c) For $0 \le i < oLen - 1$, consider the octet O_i to be the bits $b_{8i}\ b_{8i+1}\ \dots\ b_{8i+7}$.

Algorithm 10 – ROS2BSP

d) If any of the bits $b_{bLen-1} \ldots b_{8*oLen-1}$ are non-zero, output "non-zero bits found after end of bit string" and stop.

e) Output the bit string $B = b_0\, b_1 \ldots b_{bLen\text{-}1}$.

7.5 Converting Between Ring Elements and Octet Strings (RE2OSP and OS2REP)

This clause gives the primitives for converting between ring elements and octet strings.

7.5.1 Ring Element to Octet String Primitive (RE2OSP)

RE2OSP converts a ring element to an octet string.

Algorithm 11 – RE2OSP

Input: *a*: ring element to be converted, equal to $a_0 + a_1 X + a_2 X^2 + \ldots + a_{N\text{-}1} X^{N\text{-}1}$; *N*: dimension of ring; *q*: larger modulus: all coefficients of the ring element are between 0 and *q*-1.

Output: *O*: corresponding octet string

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

```
a) For $j$ = 0 to $N$-1:
    1) Set $A_j$ equal to the smallest positive representation of $a_j$ mod q.
    2) Set $O_j$ = I2OSP ($A_j$, ceil[$\log_{256}$ q]). If any of the calls to I2OSP output an error, output that error and stop.
b) Output the octet string $O = O_0\, O_1 \ldots O_{N\text{-}1}$.
```

NOTE—As an example, if *q*=128 and *N*=5, the polynomial

$$a[X] = 45 + 2X + 77\, X^2 + 103\, X^3 + 12\, X^4$$

is represented by the octet string 2d 02 4d 67 0c.

7.5.2 Octet String to Ring Element Primitive (OS2REP)

OS2REP converts an octet string to a ring element.

Algorithm 12 – OS2REP

Input: *O*: octet string to be converted; *N*: dimension of ring; *q*: larger modulus: all coefficients of the ring element are between 0 and *q*-1.

Output: *a*: resulting ring element, equal to $a_0 + a_1 X + a_2 X^2 + \ldots + a_{N\text{-}1} X^{N\text{-}1}$

Algorithm 12 – OS2REP

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

a) If the length of O is not equal to N * ceil[$\log_{256} q$], output “octet string incorrect length” and stop.

b) Consider O to be the series of octet strings $O = O_0\ O_1 \ldots O_{N-1}$., where each O_j is of length ceil[$\log_{256} q$] octets.

c) For j = 0 to N-1, set a_j = OS2IP (O_j). If a_j >= q or if OS2IP outputs an error, output “error”.

d) Output $a = a_0 + a_1 X + a_2 X^2 + \ldots + a_{N-1} X^{N-1}$.

7.6 Converting Between Ring Elements and Bit Strings (RE2BSP and BS2REP)

While octet string representation may be most convenient for ring element arithmetic in a microprocessor, ring elements may be more compactly stored and transmitted as bit strings. This clause provides the appropriate conversion primitives.

7.6.1 Ring Element to Bit String Primitive (RE2BSP)

RE2OSP converts a ring element to a bit string.

Algorithm 13 – RE2BSP

Input: a: ring element to be converted, equal to $a_0 + a_1 X + a_2 X^2 + \ldots + a_{N-1} X^{N-1}$; N: dimension of ring; q: larger modulus: all coefficients of the ring element are between 0 and q-1.

Output: B: resulting bit string.

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

a) For j = 0 to N-1:

b) Set A_j equal to the smallest positive representation of a_j mod q.

c) Set B_j = I2BSP (A_j, ceil[$\log_2$ q]). If any of the calls to I2BSP output an error, output that error and stop.

d) Output the bit string $B = B_0\ B_1 \ldots B_{N-1}$.

NOTE—As an example, if q=128 and N=5, the polynomial

$a[X] = 45 + 2X + 77 X^2 + 103 X^3 + 12 X^4$

is represented by the bit string 0101101 0000010 1001101 1100111 0001010. (If this were subsequently to be converted to an octet string using BS2ROSP, it would become first the bit string 0101 1010 0000 1010 0110 1110 0111 0001 0100 0000, and then the octet string 5a 0a 6e 71 40).

7.6.2 Bit String to Ring Element Primitive (BS2REP)

BS2REP converts a bit string to a ring element.

Algorithm 14 – BS2REP

Input: *B*: bit string to be converted; *N*: dimension of ring; *q*: larger modulus: all coefficients of the ring element are between 0 and *q*-1.

Output: *a*: resulting ring element, equal to $a_0 + a_1 X + a_2 X^2 + \ldots + a_{N-1} X^{N-1}$

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

a) If the length of *B* is not equal to *N* * ceil[$\log_2 q$], output “bit string incorrect length” and stop.

b) Consider *B* to be the series of bit strings $B = B_0 B_1 \ldots B_{N-1}$., where each B_j is of length ceil[$\log_2 q$] bits.

c) For $j = 0$ to N-1, set a_j = BS2IP (B_j). If BS2IP outputs an error, output “error”.

d) Output $a = a_0 + a_1 X + a_2 X^2 + \ldots + a_{N-1} X^{N-1}$.

8. Supporting algorithms

8.1 Overview

In order to perform the operations securely, implementers shall choose supporting algorithms that satisfy the security needs of the schemes. The security level of the supporting algorithm typically depends on the desired security level of the scheme (e.g. for a desired security level of 80 bits, the SHA-1 hash algorithm is typically chosen). This clause defines the algorithms that shall be used to meet this standard.

8.2 Hash Functions

Hash functions are used in two distinct situations in this standard: as the core of a mask generation function, and as the core of a pseudo-random bit generator. For security purposes, the hash function should be chosen at a strength commensurate to the desired security level. The recommended parameter sets in this document specify hash functions appropriate to their security levels.

The only currently supported hash functions for use within this standard are SHA-1 and SHA-256 [FIP95, NIST-SHA-2].

All hash functions in this standard take an octet string as an input and produce an octet string as an output. For compatibility with other standards which specify input and output as bit strings, the conversion primitives ROS2BSP and BS2ROSP (clauses 7.4.1 and 7.4.2) may be used.

8.3 Encoding Methods

Before a message is encrypted, it must be processed to guarantee certain desirable security properties such as semantic security. In this clause, the auxiliary methods for manipulating data for the encryption scheme are listed. These currently consist of specific methods for generating the blinding polynomial *r*.

8.3.1 Blinding Polynomial Generation Methods (BPGM)

In order to provide plaintext awareness, a blinding polynomial generation method (BPGM) shall be used to generate a blinding polynomial *r* from the padded message *pm*. This clause contains two BPGMs. The first utilizes the standard polynomial convolution method, and the second utilizes the optimized polynomial convolution method.

8.3.1.1 lbp-bpgm-3

The blinding polynomial *r* shall be generated deterministically from the message *m* and the random value *b* using a pseudo-random number generator.

Algorithm 15 – Blinding Polynomial Generation From *dr*

Components: The parameters *N* and *dr*, the chosen index generation function IGF(), the hash function Hash() chosen to parameterize IGF(), the polynomial index generation constant *c*, and the minimum number of hash calls for the IGF to make, *minCallsR*.

Input: The seed, which is an octet string *seed*

Output: The blinding polynomial, which is a polynomial *r*.

Operation: The blinding polynomial shall be computed by the following or an equivalent sequence of steps:

```
a)  Call the IGF with hash function Hash() and input seed, N, c, minCallsR to obtain the IGF state s.
b)  Set $r := 0$
c)  Set $t := 0$
d)  While $t < dr$ do
    1)  Call the IGF with input s to obtain an integer $i$ mod $N$.
    2)  If $r_i = 0$
        i)   Set $r_i := 1$
        ii)  Set $t := t + 1$
e)  Set $t := 0$
f)  While $t < dr$ do
    1)  Call the IGF with input s to obtain an integer $i$ mod $N$ and the updated state $s$. If the IGF outputs “error”, output “error”.
    2)  If $r_i = 0$
        i)   Set $r_i := -1$
        ii)  Set $t := t + 1$
g)  Return $r$
```

8.4 Supporting Algorithms

In order to perform the operations securely, implementers shall choose supporting algorithms that satisfy the security needs of the schemes. The security level of the supporting algorithm typically depends on the desired security level of the scheme (e.g. for a desired security level of 80 bits, the SHA-1 hash algorithm is typically chosen). This clause defines the algorithms that shall be used to meet this standard.

8.4.1 Mask Generation Functions

Mask Generation Functions (MGFs) are functions similar to hash functions, except that instead of producing a fixed-length output they produce an output of arbitrary length.

All mask generation functions are parameterized by the choice of a core hash function. The only hash functions supported for use with the MGFs in this standard are SHA-1 and SHA-256 [FIP95, NIST-SHA-2].

This standard only permits the use of one mask generation function, MGF-TP-1. This function takes as input an octet string and the desired degree of the output, and produces a trinary polynomial of the appropriate degree. The only hash functions supported for use with this mask generation function are SHA-1 and SHA-256 [FIP95, NIST-SHA-2].

8.4.1.1 Mask Generation Function for Trinary Polynomials (MGF-TP-1)

Algorithm 16 – Mask Generation Function for Trinary Polynomials (MGF-TP-1)

Components: A hash function *Hash* with output length *hLen* octets.

Input: an octet string *seed* of length *seedLen* octets; the degree *N*, an integer; an argument *hashSeed*, taking the values "yes" or "no"; and the minimum number of calls *minCallsMask*, an integer

Output: An polynomial *i* of degree *N*-1; or “error”.

Operation: The integer and state shall be produced by the following or an equivalent sequence of steps:

a) If *seedLen*+4 exceeds any input length limitation on the hash function *Hash*, output “error” and exit

b) If *minCallsMask* exceeds 2^{32}, output “error” and exit.

c) Check the value of *hashSeed*.

   1) If *hashSeed* = "yes", set the octet string *Z* to *Hash*(*seed*) and the integer *zLen* to *hLen*.

   2) If *hashSeed* = "no", set the octet string *Z* to *seed* and the integer *zLen* to *seedLen*.

d) Initialize the octet string *buf* to be a zero-length octet string.

e) Initialize *counter*:= 0.

f) Initialize *N* and *c* with the provided values. Set *cLen* = ceil (c/8).

g) While *counter* < *minCallsR* do

   1) Convert *counter* to an octet string *C* of length 4 octets using I2OSP.

2) Compute *Hash*(Z || C) with the selected hash function to produce an octet string *H* of length *hLen* octets.
3) Let $buf = buf \| H$.
4) Increment *counter* by one.

h) Initialize *i* to be the null polynomial and *cur*, a pointer to the current coefficient of *i*, to be 0.

i) For each octet *o* in *buf:*
   1) Convert *o* to an integer *O*.
   2) If $O >= 243$ ($= 3^5$) discard *O*, move to the next octet, and go to step d)1).
   3) Set $i_{cur} = O \bmod 3$; if $cur = N$ output *i*; set $cur = cur + 1$; set $O = (O - O \bmod 3) / 3$.
   4) Set $i_{cur} = O \bmod 3$; if $cur = N$ output *i*; set $cur = cur + 1$; set $O = (O - O \bmod 3) / 3$.
   5) Set $i_{cur} = O \bmod 3$; if $cur = N$ output *i*; set $cur = cur + 1$; set $O = (O - O \bmod 3) / 3$.
   6) Set $i_{cur} = O \bmod 3$; if $cur = N$ output *i*; set $cur = cur + 1$; set $O = (O - O \bmod 3) / 3$.
   7) Set $i_{cur} = O$; if $cur = N$ output *i*; set $cur = cur + 1$

j) If $cur < N$:
   1) Convert *counter* to an octet string *C* of length 4 octets using I2OSP.
   2) Compute *Hash*(Z || C) with the selected hash function to produce an octet string *H* of length *hLen* octets.
   3) Let $buf = H$.
   4) Increment *counter* by one.
   5) return to step i).

k) Output *i*.

8.4.2 Index generation function

The term “index generation function”, as used in this standard, applies to functions which are initialized with a seed in the form of an octet string and may then be called repeatedly, producing an integer in a specified range on each call.

An IGF may be deterministic or non-deterministic. A deterministic IGF is parameterized by a hash function; the only hash functions supported for use with the IGFs in this standard are SHA-1, SHA-256, SHA-384, and SHA-512. On initialization, it takes as input a seed, which is an octet string; a modulus N; an index generation constant c; and the desired minimum number of calls to the underlying hash function, minCallsR. It outputs an integer in the range [0, N-1] and the internal state *s*. On subsequent calls, it takes as input the current state *s* and outputs an octet string of length *oLen* and the updated internal state *s*.

This standard permits the use of a deterministic index generation function based on a hash function and a nondeterministic index generation function based on a random bit generator.

8.4.2.1 Index generation function (IGF-2)

Algorithm 17 – Index generation function (IGF-2)

Components: A hash function *Hash* with output length *hLen* octets.

Input:

EITHER: an octet string *seed* of length *seedLen* octets; the modulus *N*, an integer; an argument *hashSeed*, taking the values "yes" or "no"; the index generation constant *c*, an integer; and the minimum number of calls *minCallsR*, an integer

OR: the state *s*.

Output: An integer *i* and the state *s*; or “error”.

Operation: The integer and state shall be produced by the following or an equivalent sequence of steps:

a) If *s* is not provided:
   1) If *seedLen*+4 exceeds any input length limitation on the hash function *Hash*, output “error” and exit
   2) If *minCallsR* exceeds 2^{32}, output “error” and exit.
   3) Check the value of *hashSeed*.
      i) If *hashSeed* = "yes", set the octet string *Z* to *Hash*(*seed*) and the integer *zLen* to *hLen*.
      ii) If *hashSeed* = "no", set the octet string *Z* to *seed* and the integer *zLen* to *seedLen*.
   4) Initialize *totLen* to 0. Intialize *remLen* to 0.
   5) Initialize the bit string buf to be a zero-length bit string.
   6) Initialize *counter*:= 0.
   7) Initialize *N* and *c* with the provided values.
   8) While *counter* < *minCallsR* do
      i) Convert *counter* to an octet string *C* of length 4 octets using I2OSP.
      ii) Compute *Hash*(Z || C) with the selected hash function to produce an octet string *H* of length *hLen* octets.
      iii) Let *buf* = *buf* || OS2BSP(*H*).
      iv) Increment *counter* by one.
   9) Set *remLen* = *totLen* = *minCallsR* * 8**hLen*.

b) Otherwise (if *s* is provided):
   1) Extract the values *Z*, *totLen*, *remLen*, *buf*, *counter*, *N*, *c* from the state *s*. (The details of how they are stored in *s* may be determined by the implementer).

c) Set totLen:=totLen + c.

d) If totLen exceeds hLen $\times 8 \times 2^{32}$, output “error” and exit.

Algorithm 17 – Index generation function (IGF-2)

```
e) If remLen < c
    1) Let the bit string $M$ be the trailing $remLen$ bits in $buf$.
    1) Let $tmpLen := c – remLen$.
    2) Let $cThreshold = counter + \text{ceil}[tmpLen/hLen]$.
    3) While $counter < cThreshold$ do
        i) Convert $counter$ to an octet string $C$ of length 4 octets using I2OSP.
        ii) Compute $Hash(Z || C)$ with the selected hash function to produce an octet string
            $H$ of length $hLen$ octets.
        iii) Let $M = M || \text{OS2BSP}(H)$.
        iv) Increment $counter$ by one. If $tmpLen > 8*hLen$, decrement $tmpLen$ by $8*hLen$.
    4) Set $remLen := 8*hLen – tmpLen$. Set $buf := H$.
f) else
    1) Set $M$ equal to the trailing $remLen$ bits of $buf$.
    2) Set $remLen := remLen – c$.
g) Set the bit string $b$ to the leading $c$ bits in $M$,
h) Convert $b$ to an integer $i$ using OS2IP.
i) If $i >= 2^c - (2^c \bmod N)$ go back to step 3.
j) Store the values $Z$, $totLen$, $remLen$, $counter$, $N$, $cLen$ and $c$ in the state $s$. (The details of
    how they are stored in $s$ may be determined by the implementer).
k) Output $i \bmod N$ and $s$.
```

8.4.2.2 Index generation function (IGF-RBG)

This IGF is based on any approved random bit generator

Algorithm 18 – Index generation function (IGF-RBG)

Components: An Approved random bit generator RBG

Input: The modulus N, an integer; the index generation constant c, an integer.

Output: An integer i

Operation: The integer i shall be produced by the following or an equivalent sequence of steps:

```
1. Set $cLen = \text{ceil}(c/8)$.
2. Obtain a bit string $b$ of length $8*cLen$ bits from RBG.
3. Convert $b$ to an octet string $o$ using BS2OSP.
4. Set the leftmost $8cLen - c$ bits of $o$ to 0.
5. Convert $o$ to an integer $i$ using OS2IP.
6. If $i >= 2^c - (2^c \bmod N)$ go back to step 3.
```

Algorithm 18 – Index generation function (IGF-RBG)

```
7.  Output $i$ mod $N$.
```

9. Short Vector Encryption Scheme (SVES)

The following clause defines the supported encryption schemes. The only encryption scheme currently supported is SVES. SVES stands for Short Vector Encryption Scheme (see for more information).

9.1 Encryption Scheme (SVES) Overview

The general encryption scheme is a sequence of operations that are performed based on the choices of the parameters, primitives, encoding functions and supporting algorithms. In order to perform all of the SVES encryption scheme operations, all of the Components must be specified.

9.2 Encryption Scheme (SVES) Operations

The SVES encryption scheme consists of the five operations key generation, key pair validation, public key validation, encryption and decryption. These operations are defined generally in this clause without assuming any specific choices of the Components listed in Clause 9.1 Encryption Scheme (SVES) Overview.

9.2.1 Key Generation

A key pair shall be generated using the following or a mathematically equivalent set of steps. Note that the algorithm below outputs only the values *f* and *h*. In some applications it may be desirable to store the values f^{-1} and *g* as well. This standard does not specify the output format for the key as long as it is unambiguous

Algorithm 19 – Random Key Generation Primitive kgp-3

Components: The parameters *N, q, p, dF, dg*; EITHER an Approved random number generator capable of generating unbiased output in the range (0, N-1) OR an index generation function IGF that takes an Approved random bit generator RBG and the polynomial index generation constant *c* used by the IGF.

Input: None

Output: An key pair consisting of the private key *f* and the public key *h*

Operation: The key pair shall be computed by the following or an equivalent sequence of steps:

```
a)  Set the polynomial $F := 0$.
b)  Set $t := 0$
c)  While $t < dF$ do
    1)  Call EITHER the RNG OR the IGF with input N, c, RBG to obtain an integer $i$ ,mod $N$.
```

```
        2)  If $F_i = 0$
            i)   Set $F_i := 1$
            ii)  Set $t := t + 1$
    d)  Set t:=0 While $t < dF$ do
        1)  Call EITHER the RNG OR the IGF with input N, c, RBG to obtain an integer $i$ mod $N$.
        2)  If $F_i = 0$
            i)   Set $F_i := -1$
            ii)  Set $t := t + 1$
    e)  Compute the polynomial $f := 1 + p*F$ in $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$
    f)  Compute the polynomial $f^{-1}$ (i.e. the polynomial $f^{-1}$ such that $f^{-1}*f = f*f^{-1} = 1$) in $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$.  If $f^{-1}$ does not exist, go to step 1.
    g)  Set the polynomial $g := 0$.
    h)  Set $t := 0$
    i)  While $t < dg$ do
        1)  Call EITHER the RNG OR the IGF with input N, c, RBG to obtain an integer $i$ mod $N$.
        2)  If $g_i = 0$
            i)   Set $g_i := 1$
            ii)  Set $t := t + 1$
    j)  Set $t := 0$
    k)  While $t < dg$ do
        1)  Call EITHER the RNG OR the IGF with input N, c, RBG to obtain an integer $i$ mod $N$.
        2)  If $g_i = 0$
            i)   Set $g_i := -1$
            ii)  Set $t := t + 1$
    l)  Check that $g$ is invertible mod $q$. If it is not, go back to step 8.
    m)  Compute the polynomial $h := f^{-1}*g*p$ in $(\mathbf{Z}/q\mathbf{Z})[X]/(X^N - 1)$
    n)  Output $f$, $h$
```

9.2.2 Encryption Operation

This clause defines the Encryption operation. Note that within the definition of the spaces may be definitions of additional variables (e.g. when defining D_r, the values $dr1$, $dr2$ and $dr3$ may be specified as well as the appropriate method of combining them).

Algorithm 20 – Encryption Operation

Algorithm 20 – Encryption Operation

Components:

- The length of the encoded length *lLen*.
- The number of bits of random data *db*, which must be a multiple of 8.
- The chosen Mask Generation Function and associated parameters.
- The chosen Blinding Polynomial Generation Method and the associated parameters
- The OID, an octet string
- The number of bits of public key to hash, *pkLen*.
- The minimum message representative weight, *dm0*.
- The minimum number of calls to generate the masking polynomial, *minCallsMask*.
- The maximum message length *maxMsgLenBytes*
- The minimum number of calls to generate the blinding polynomial, *minCallsR*.
- The length of the encoding buffer, *bufferLenBits*

Inputs:

- The message *m*, which is an octet string of length *l* octets
- The public key *h*

Output: The ciphertext *e*, which is a ring element, or "message too long"

Operation: The ciphertext *e* shall be calculated by the following or an equivalent sequence of steps:

a) Calculate *octL* = the *lLen*-octet-long encoding of the message length *l*.

b) If *l* > *maxLen*, output "message too long" and stop.

c) Randomly select an octet string *b* of length *bLen* using a random number generator with at least 8**bLen* bits of entropy content.

d) Form the octet string *p0*, consisting of the 0 byte repeated (*maxMsgLenBytes* + 1 - *l*) times.

e) Form the octet string *M* of length *bufferLenBits*/8 as *b* || *octL* || *m* || *p0*.

f) Convert *M* to a bit string *Mbin* using OS2BSP.

g) If *Mbin* is not a multiple of three bits long, append 0 bits to bring it up to a multiple of three.

h) Convert Mbin to a trinary polynomial of degree N-1 as follows. Treat Mbin as a concatenation of 3-bit quantities. Convert each three-bit quantity to two trinary coefficients as follows, and concatenate the resulting trinary quantities to obtain Mtrin.

  - {0, 0, 0} -> {0, 0}
  - {0, 0, 1} -> {0, 1}
  - {0, 1, 0} -> {0, -1}
  - {0, 1, 1} -> {1, 0}

Algorithm 20 – Encryption Operation

— {1, 0, 0} -> {1, 1}

— {1, 0, 1} -> {1, -1}

— {1, 1, 0} -> {-1, 0}

— {1, 1, 1} -> {-1, 1}

i) Convert the public key *h* to a bit string *bh* using RE2BSP (7.6.1). Form the bit string *bhTrunc* by taking the first *pkLen* bits of *bh*. Convert *bhTrunc* to the octet string *hTrunc*, of length *pkLen*/8 using BS2OSP. Form *sData* as the octet string
OID || *m* || *b* || *hTrunc*

j) Use the chosen blinding polynomial generation method with the seed *sData* and the chosen parameters to produce *r*. IF the blinding polynomial generation method outputs “error”, output “error”.

k) Calculate $R = r*h \bmod q$.

l) Calculate $R4 = R \bmod 4$.

m) Convert *R4* to the octet string *oR4* using BE2OSP.

n) Generate a masking polynomial *mask* by calling the given MGF with inputs (*oR4*, *N*, *minCallsMask*)

o) Form m' by polynomial addition of *M* and *mask* mod p.

p) If the number of 1s, or -1s, or 0s in m’ is less than dm0, discard m’ and return to step 3.

q) Calculate the ciphertext as e = R + m’ mod q.

r) Output e.

Graphically, the encryption operation may be represented as follows:

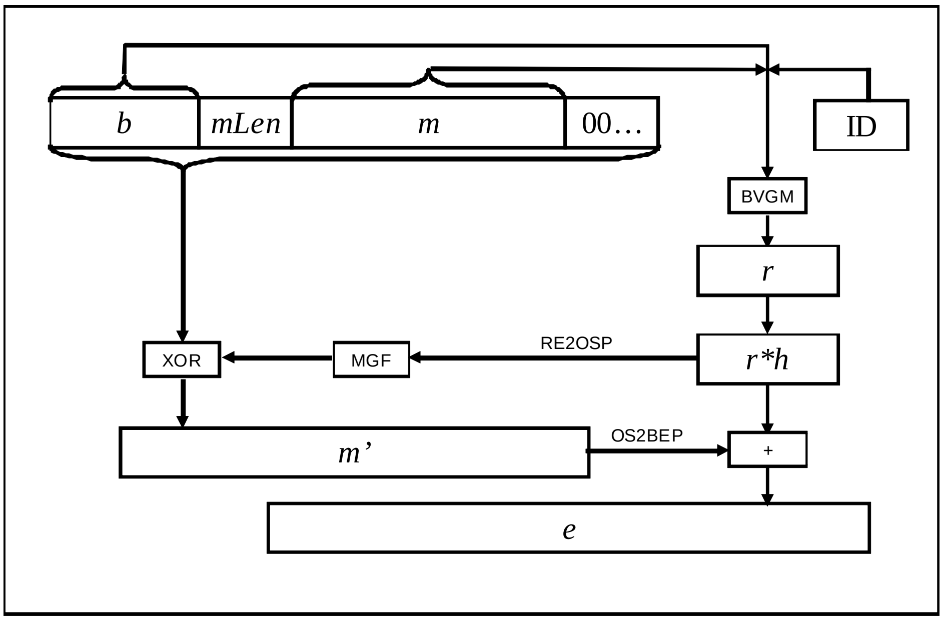

Figure 1: Encryption Operation

9.2.3 Decryption Operation

This clause defines the decryption operation. Note that within the definition of the spaces may be definitions of additional variables (e.g. when defining D_r, the values $dr1$, $dr2$ and $dr3$ may be specified as well as the appropriate method of combining them).

Algorithm 21 – Decryption Operation

Components:

- The LBP-PKE decryption primitive to use
- The length of the encoded length *lLen*.
- The number of bits of random data *db*, which must be a multiple of 8.
- The chosen Mask Generation Function and Hash Function.
- The chosen Blinding Polynomial Generation Method and the associated parameters
- The OID, an octet string
- The number of bits of public key to hash, *pkLen*.
- The lower bound *A*
- The minimum message representative weight *dm*0
- The maximum message length *maxMsgLenBytes*

Inputs:

- The ciphertext *e*, which is a polynomial of degree N-1.
- The private key *f* or (f, f_p).
- The public key *h*

Output: The message *m*, which is an octet string, or "fail".

Operation: The message m shall be calculated by the following or an equivalent sequence of steps:

a) Calculate:
   1) *nLen* = ceil [*N*/8], the number of octets required to hold *N* bits.
   2) *bLen* = *db*/8, the length in octets of the random data
   3) *maxLen* = *nLen* - 1 - *lLen* - *bLen*, the maximum message length.

b) Decrypt the ciphertext *e* using the selected NTRU decryption primitive with inputs *e* and *f* to get the candidate decrypted polynomial *ci*.

c) If the number of 1s, or -1s, or 0s in *ci* is less than dm0, set “fail” to 1.

d) Calculate the candidate value for *r*h*, *cR* = *e* - *ci*.

e) Calculate *cR4* = *cR* mod 4.

f) Convert *cR4* to the octet string *coR4* using BE2OSP.

g) Generate a masking polynomial *mask* by calling the given MGF with inputs (*coR4*, *N*,

Algorithm 21 – Decryption Operation

minCallsMask)

h) Form cMtrin by polynomial subtraction of *cm'* and *mask* mod p.

i) Convert cMtrin to a bit string as follows. Treat cMtrin as a concatenation of polynomials each containing 2 trinary coefficients. Convert each set of two trinary coefficients to three bits as follows, and concatenate the resulting bit quantities to obtain cMbin

- {0, 0} -> {0, 0, 0}
- {0, 1} -> {0, 0, 1}
- {0, -1} -> {0, 1, 0}
- {1, 0} -> {0, 1, 1}
- {1, 1} -> {1, 0, 0}
- {1, -1} -> {1, 0, 1}
- {-1, 0} -> {1, 1, 0}
- {-1, 1} -> {1, 1, 1}
- {-1, -1} -> set "fail" to 1 and set bit string to {1, 1, 1}

j) If cMbin is not a multiple of 8 bits long, remove the final (length – length mod 8) bits.

k) Convert c*Mbin* to an octet string *cM* using BS2OSP.

l) Parse *cM* as follows.

m) The first *bLen* octets are the octet string *cb*.

n) The next *lLen* octets represent the message length. Convert the value stored in these octets to the candidate message length *cl*. If *cl* > *maxMsgLenBytes*, set *fail* = 1 and set *cl* = *maxL*.

o) The next *cl* octets are the candidate message *cm*. the remaining octets should be 0. If they are not, set fail = 1.

p) Convert the public key *h* to a bit string *bh* using RE2BSP (7.6.1). Form the bit string *bhTrunc* by taking the first *pkLen* bits of *bh*. Convert *bhTrunc* to the octet string *hTrunc*, of length *pkLen*/8 using BS2OSP. Form *sData* as the octet string
OID || *m* || *b* || *hTrunc*

q) Use the chosen blinding polynomial generation method with the seed *sData* and the chosen parameters to produce *r*.

r) Calculate *cR*' = *h* * *cr* mod *q*.

s) If *cR*' != *cR*, set *fail* = 1

t) If *fail* = 1, output "fail". Otherwise, output *cm* as the decrypted message *m*.

9.2.4 Key Pair Validation Methods

A key pair validation method determines whether a candidate LBP-PKE public-key/private-key pair meets the constraints for key pairs produced by a particular key generation method.

9.2.4.1 kpv3: Key Pair Validation for Trinary Keys

This key validation method corresponds to the key generation operation in 9.2.1.

Algorithm 22 – kpv3, Key Pair Validation for Trinary Keys

Components: The parameters *N*, *q*, *dF*, *dg*,

Input: The private key component *F* and the public key *h*.

Output: “valid” or “invalid”.

Operation:

a) Check that *F* and *h* are polynomials of degree no greater than *N*-1. If either of them has greater degree, output “invalid” and stop.

b) Check that all of the coefficients of *h* lie in the range [0, q-1]. If any coefficients lie outside this range, output “invalid” and stop.

c) Check that *F* is trinary with exactly *dF* 1s and <u>*dF*</u> -1s. If it is not, output “invalid” and stop.

d) Set $f = 1 + 3F \bmod q$.

e) Set $g = f*h*3^{-1} \bmod q$.

f) Check that *g* is trinary with exactly *dg* 1s and *dg* -1s. If it is not, output “invalid” and stop.

g) Output “valid”.

9.2.5 Public-key validation

9.2.5.1 Full public-key validation

A full public-key validation method determines whether a candidate public key satisfies the definition of a public key and meets any additional constraints imposed by a given key pair generator. Such methods provide the highest assurance to a relying party. For example, for keys generated using the key generation operation in 9.2.1, full public-key validation would prove that $h = f^{-1}g \bmod q$, where $f = 1+pF$ and *F*, *g* have *dF*, *dg* 1s respectively. Currently there are no known methods that provide full public-key validation for the LBE-PKE schemes in this standard.

9.2.5.2 Partial public-key validation and plausibility tests

9.2.5.2.1 Overview

A partial public-key validation method determines, with some level of assurance, whether a candidate public key meets *some* of the properties of a public key. As with full public-key validation methods, partial public-key validation methods may be interactive or non-interactive. This standard supports only non-interactive methods.

Non-interactive methods for LBP-PKE public keys that do not require a witness are called *plausibility tests*. The name reflects the fact that while examining only the public key, the tests only determine whether the public key is plausible, not necessarily whether it is valid. Plausibility tests can detect unintentional errors with reasonable probability, though not with certainty. (See Note.)

This is still an active research area; further methods may be described in future versions of this Standard.

NOTE—There are other ways to detect unintentional errors; a checksum on the key will detect storage and transmission errors, and the signature on a certificate will likely fail if the public key is modified. The checks in this clause provide an additional level of assurance beyond the other methods, or an alternative when they are not available.

9.2.5.2.2 Example suite of plausibility tests

The following is an example of a plausibility test, corresponding to the the key generation operation in 9.2.1.

a) Check that $h(1) = g(1)/(1 + pF(1)) \bmod q$. (For binary polynomials, $F(1) = dF$; for product-form polynomials, $F(1) = df1*df2+df3$. In both cases, $g(1) = dg$). If it is not, output “invalid” and stop.

b) For $t = 0$ to q-1:

    1) Reduce h into the range $[t, t+q-1]$.

c) Calculate the centered norm $||h||$ for h reduced into this range.

d) Set $||h||_{min}$ equal to the minimum value of $||h||$ obtained in the previous step.

e) Set $||r|| = \sqrt{\ [\ r(1)*(N-r(1))/N\ ]}$. (For binary polynomials, $r(1) = dr$; for product-form polynomials, $r(1) = dr1*dr2+dr3$).

f) If $||h||_{min} > q\ (\sqrt{N})\ /\ (3\ ||r||)$, output “plausible public key” and stop. Otherwise, output “invalid” and stop.

Steps 2-4 are motivated by the considerations of A.4.2: for a valid public key h, the calculation of $h*r \bmod q$ will involve a large number of reductions mod q. The test checks that $||h*r|| > q\sqrt{(N)}/2$, in other words that the centered norm of h*r is with high likelihood greater than the centered norm of a polynomial consisting of $N/2$ coefficients with the value $q/2$ and $N/2$ coefficients with the value $–q/2$ (this calculation uses the pseudo-multiplicative property of the centered norm defined in A.1.1). For genuine h, the typical value of $||h||_{min}$ will be slightly under $q\ \sqrt{(N/12)}$. For binary polynomials, the centered norm $||r||$ will be $\sqrt{(dr\ (N-dr)/N)}$, which is considerably greater than $\sqrt{(3)}$ for all parameter sets in this standard. A valid h will therefore pass this test with high probability. For product-form polynomials, the value of $||r||$ will vary, but its minimum value will be $\sqrt{(d\ (N-d)/N)}$, where $d= dr1*dr2+dr3$. This will also be considerably greater than $\sqrt{(3)}$ for all parameter sets in this standard, and a valid h will pass this test with high probability.

Annex A (Informative) Security Considerations

A.1 Lattice Security: Background

This section provides an overview of the properties of lattices, as a necessary preliminary to considering the security of cryptosystems based on hard lattice problems.

A.1.1 Lattice Definitions

A *lattice* is of dimension n is a maximal discrete subgroup of real n-dimensional space $\mathbf{R}^n$. A lattice L may be specified by a spanning set of n linearly independent vectors $\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_n\}$ called a *basis for* L, in which case L is the set of vectors

$$L = \{ x_1\boldsymbol{b}_1 + \ldots + x_n\boldsymbol{b}_n : x_1,\ldots,x_n \in \mathbf{Z} \}.$$

A lattice has many bases. A lattice is called *integral* if it is contained in $\mathbf{Z}^n$ and it is called *rational* if it is contained in $\mathbf{Q}^n$. A (*row*) *matrix* for L is a matrix whose rows form a basis for L. The *discriminant of* L, denoted Disc(L), is the determinant of any matrix for L; the value is independent of the choice of basis. The discriminant is also characterized as the volume of a fundamental domain for the quotient space $\mathbf{R}^n/L$, so it is also sometimes called the *volume* (really co-volume) *of* L.

The *L^2-norm* and the *centered L^2-norm* of a vector $\boldsymbol{a}$ are given by the respective formulas

$$\| a(X) \|_2 = \sqrt{\sum_{i=0}^{N-1} a_i^2} \quad \text{and} \quad \| a(X) \|_{2,ctr} = \sqrt{\sum_{i=0}^{N-1} a_i^2 - \frac{1}{N}\left(\sum_{i=0}^{N-1} a_i\right)^2} \quad .$$

Let $\boldsymbol{a}_{\text{avg}}$ be the vector whose coordinates are all equal to $(a_0+a_1+\ldots+a_{N-1})/N$, the average of the coordinates of $\boldsymbol{a}$. Then the centered L^2-norm of $\boldsymbol{a}$ may also be defined by $\|\boldsymbol{a}\|_{2,\text{ctr}} = \|\boldsymbol{a} - \boldsymbol{a}_{\text{avg}}\|_2$.

A vector $\boldsymbol{a}$ is said to be *centered* if $a_0+a_1+\ldots+a_{N-1} = 0$, that is, if the average of its coordinates is 0. (If the vectors a and b represent polynomials, the sum $a(X)+b(X)$ and the product $a(X)*b(X)$ of centered polynomials $a(X)$ and $b(X)$ are themselves centered).

The L^2-norm of the (convolution) product of two independent centered polynomials $a(X)$ and $b(X)$ may be estimated by the formula

$$\|a(X) * b(X)\|_2 \approx \|a(X)\|_2 * \|b(X)\|_2.$$

This is known as the *pseudo-multiplicative property* of the centered norm.

The *first minimum of* L, denoted $\lambda(L)$ or $\lambda_1(L)$, is the length of the smallest nonzero vector in L. More generally, for each $1 \le i \le n$, the i^{th} *successive minimum of* L, denoted $\lambda_i(L)$, is the infimum of all numbers λ such that L contains i linearly independent vectors of length at most λ. *Hermite's constant* γ_n is the infimum of the ratio $\lambda_1(L)^2/\text{Disc}(L)^{2/n}$ as L runs over all lattices of dimension n. It is known that $\gamma_n\ \theta(n)$, although the exact value of γ_n is only known for $1 \le n \le 8$.

Let $\boldsymbol{a} \in \mathbf{R}^n$. The distance from $\boldsymbol{a}$ to L, denoted $\lambda(L,\boldsymbol{a})$, is the distance from $\boldsymbol{a}$ to the closest vector in L.

A.1.2 Hard Lattice Problems

The *shortest vector problem* (SVP) for a lattice L is to find a vector $\boldsymbol{v} \varepsilon L$ satisfying $\|\boldsymbol{v}\| = \lambda_1(L)$, that is, to find a vector of shortest nonzero length. The *approximate short vector problem* (apprSVP) is to find a vector $\boldsymbol{v} \varepsilon L$ satisfying $\|\boldsymbol{v}\| \le f(n)\lambda_1(L)$ for some (slowly growing) function f of the dimension n.

The *closest vector problem* (CVP) for a lattice L and vector $\boldsymbol{a} \varepsilon \mathbf{R}^n$ is the problem of finding a vector $\boldsymbol{v} \varepsilon L$ satisfying $\|\boldsymbol{v} - \boldsymbol{a}\| = \lambda(L,\boldsymbol{a})$, i.e., minimizing the distance $\|\boldsymbol{v} - \boldsymbol{a}\|$. The *approximate closest vector problem* (apprCVP) is to find a vector $\boldsymbol{v} \varepsilon L$ satisfying $\|\boldsymbol{v} - \boldsymbol{a}\| \le f(n)\lambda(L,\boldsymbol{a})$ for some (slowly growing) function f of the dimension n.

The *smallest basis problem* (SBP) for a lattice L has many different formulations depending on how one measures the “smallness” of a basis. A common definition is to minimize the length of the longest element of the basis. Another common definition is to minimize the product of the lengths of the elements in the basis.

A.1.3 Theoretical Complexity of Hard Lattice Problems

It is known that SVP is NP-hard under randomized reductions Annex B, and the same is true for apprSVP with approximating factor $\sqrt{2}$ [B75]. It is known that CVP is NP-hard [B21]. Although CVP appears to be somewhat harder than SVP, it is known that an algorithm to solve apprSVP with approximating function $f(n)$ can be used to solve apprCVP with approximating function $n^{3/2}f(n)$ [B60], so the two are polynomially equivalent. In practice, a CVP in dimension n can often be solved by transforming it into an SVP in dimension n+1. In the other direction, it is very unlikely that apprSVP or apprCVP is NP-hard for the approximating function $f(n) \approx (n/\log n)^{1/2}$ [B24].

A.1.4 Lattice Reduction Algorithms

Let L be an integral (or rational) lattice of dimension n. An exhaustive search can be used to solve SVP or CVP, with expected running time exponential in n. There are algorithms for solving apprSVP and apprCVP with polynomial (in n) running time and (slightly better than) exponential approximating factor $f(n)$. More precisely, the LLL algorithm [B69] runs in polynomial time and is guaranteed to return a nonzero vector $\boldsymbol{v} \varepsilon L$ satisfying $\|\boldsymbol{v}\| \le 2^{n/2}\lambda_1(L)$; the approximating factor can be improved to $2^{O(n(\log\log n)^2/\log n)}$ [B89]. More generally, [B89][B90][B91] describe block variants of the LLL algorithm called BKZ-LLL whose running time and approximating factor depend on the choice of a block size β. Larger values of β lead to better results and longer running times. The BKZ-LLL algorithm with block size β is guaranteed to find a nonzero vector $\boldsymbol{v} \varepsilon L$ satisfying

$$\|\boldsymbol{v}\| \le (2.45\beta)^{n/\beta}\,\lambda_1(L) \quad \text{in time at most} \quad O(n^2(\beta^{\beta/2+o(\beta)} + n^2)).$$

Thus in order to obtain a provable polynomial approximation factor, the block size β must be proportional to the dimension n, in which case the running time is (at least) exponential in the dimension.

In practice, the LLL algorithm and its BKZ-LLL variants tend to return answers that are somewhat better than the upper bounds given by theory. However, also in practice, the shortest vector returned by BKZ-LLL tends to be considerably longer than $\lambda_1(L)$ until the block size β is an appreciable fraction of the dimension n. Also in practice, the running time of BKZ-LLL is (at least) exponential in the block size β. In other words, even in practice, BKZ-LLL is unlikely to find a vector as short as $c^{n/\beta}$ in time less than $O(n^2\beta^{\beta/2})$.

Recent research [B92] suggests another block-based algorithm known as Random Sampling Reduction (RSR), which is guaranteed to find a nonzero vector $\boldsymbol{v} \varepsilon L$ satisfying

$$\|\boldsymbol{v}\| \le (k/6)^{n/2k}\,\lambda_1(L) \quad \text{in time approximately} \quad O(n^3(k/6)^{k/4}).$$

For exact solutions to SVP and CVP, there are superexponential algorithms [B59][B61] with running time $2^{O(n \log n)}$ and a more recent algorithm with exponential running time [B3]. Other lattice reduction algorithms are described in [B68][B101][B13][B82]. The review [B39] considers known lattice attacks and concludes that no better attack is currently known than straightforward BKZ.

For solving a CVP of dimension n, the best method in practice is to embed it into an SVP of one higher dimension [B25][B80]. Let $(L,\boldsymbol{a})$ be a CVP. Then one takes a basis $\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_n\}$ for L, forms the lattice L^* in $\mathbf{R}^{n+1}$ with basis $\{[\boldsymbol{b}_1,0],\ldots,[\boldsymbol{b}_n,0],[\boldsymbol{a},c]\}$ for an appropriate constant c and hopes that a shortest vector in L^* has the form $[\boldsymbol{u},c]$, in which case the vector $\boldsymbol{a}+\boldsymbol{u}$ is in L and is likely to be a closest vector to $\boldsymbol{a}$.

A.1.5 The Gaussian Heuristic and the Closest Vector Problem

Let L be a lattice and let $\boldsymbol{a} \varepsilon \mathbf{R}^n$ be a vector. The *Gaussian heuristic* says that all other things being equal, the distance from $\boldsymbol{a}$ to the closest vector in L is probably approximately equal to the value of R specified by following condition:

$$\text{Volume of a ball of radius } R \text{ around } \boldsymbol{a} > \text{Discriminant of } L.$$

The intuition underlying the Gaussian heuristic is that all of $\mathbf{R}^n$ can be covered by disjoint n-dimensional parallelopipeds of volume Disc(L) centered at the points of L, so any nicely shaped region with the same volume is likely to contain a point of L. Using the formula $\pi^{n/2}/(n/2)!$ for the volume of an n dimensional ball (n even) and using Stirling's formula to approximate factorials as $k! \approx (k/e)^k(2\pi k)^{1/2}$, the Gaussian heuristic says that in a lattice of large dimension n, the critical radius is given by

$$R_{\text{crit}}(L) = (n/2\pi e)^{1/2}\,\text{Disc}(L)^{1/\dim(L)}.$$

If R is somewhat larger than $R_{\text{crit}}(L)$, then the Gaussian heuristic predicts that there will be many vectors of L that are within a distance R of $\boldsymbol{a}$; while if R is smaller than $R_{\text{crit}}(L)$, then the Gaussian heuristic predicts that there will be few or no vectors of L that are within a distance R of $\boldsymbol{a}$.

Let L be a lattice of dimension n and let $\boldsymbol{a} \varepsilon \mathbf{R}^n$. In many situations of cryptographic interest, one hides a vector $\boldsymbol{v} \varepsilon L$ that is a known (short) distance δ from the known vector $\boldsymbol{a}$. Thus the lattice L, the vector $\boldsymbol{a}$, and the distance δ are public knowledge, while the vector $\boldsymbol{v}$ is the private information. The Gaussian heuristic can be used to predict if $\boldsymbol{v}$ is likely to be a closest vector to $\boldsymbol{a}$, in which case recovery of the private information is probably equivalent to solution of the CVP for $(L,\boldsymbol{a})$. More precisely, the Gaussian heuristic says that if $\delta = \|\boldsymbol{v} - \boldsymbol{a}\|$ is significantly smaller than $(n/2\pi e)^{1/2}\,\text{Disc}(L)^{1/n}$, say less than ½ or ⅓ of this quantity, then $\boldsymbol{v}$ is probably a solution to the CVP for $(L,\boldsymbol{a})$ and $\delta = \lambda(L,\boldsymbol{a})$.

A.1.6 Modular Lattices: Definition

A *Modular Lattice* (ML) with dimension parameter $n = 2N$ and modulus parameter q is a lattice of dimension n generated by the rows of an n-by-n matrix of the form

$$\begin{bmatrix} b & \cdots & 0 & h_{11} & \cdots & h_{1N} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & b & h_{N1} & \cdots & h_{NN} \\ 0 & \cdots & 0 & q & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & q \end{bmatrix}$$

The entries of the ML matrix are integers. Without loss of generality, it may be assumed that the integers h_{ij} all satisfy $|h_{ij}| \leq q/2$, since this may be achieved by subtracting appropriate multiples of the bottom N rows from the top N rows. The integer b is called the *balancing constant*. It is selected to balance the two halves of the target vector.

It is often convenient to write an ML matrix in abbreviated form as $\begin{bmatrix} bI & h \\ 0 & qI \end{bmatrix}$, where I denotes an N-by-N identity matrix, 0 denotes an N-by-N zero matrix, and h denotes an N-by-N matrix with integer entries.

A.1.7 Modular Lattices and Quotient Polynomial Rings

It is convenient to identify a polynomial $F(X) = F_0 + F_1X + F_2X^2 + \ldots + F_{N-1}X^{N-1}$ of degree less than N with its vector of coefficients $\boldsymbol{F} = [F_0, F_1, F_2, \ldots, F_{N-1}]$. If $F(X)$ and $G(X)$ are two polynomials, let $[\boldsymbol{F}, \boldsymbol{G}]$ be the vector of dimension $2N$ formed by concatenating their coefficients.

Let $M(X)$ ε $\mathbf{Z}_q[X]$ be a monic polynomial of degree N. Then each polynomial $h(X)$ in the quotient ring $\mathbf{Z}_q[X]/(M(X))$ can be used to form a modular lattice L_h as follows:

$$L_h = \{ [\boldsymbol{F}, \boldsymbol{G}] : F(X) * h(X) = G(X) \text{ in } \mathbf{Z}_q[X]/(M(X)) \}.$$

In other words, the lattice L_h is formed from all polynomials $F(X)$,$G(X)$ ε $\mathbf{Z}[X]$ satisfying

$$F(X) * h(X) \equiv G(X) \quad (\text{modulo } q \text{ and } M(X)).$$

The i^{th} row of the N-by-N upper righthand block of the matrix for L_h is formed from the coefficients of the remainder when $X^ih(X)$ is divided by $M(X)$. In the important case that $M(X) = X^N - 1$, this block is the circulant matrix formed from the coefficients of $h(X)$ (See A.1.12).

The following procedure will create a modular lattice containing a preselected vector $[\boldsymbol{f}, \boldsymbol{g}]$. Choose $h(X)$ to satisfy $h(X) \equiv f(X)^{-1} * g(X)$ (modulo q and $M(X)$). [This assumes that $f(X)$ has an inverse in the ring $\mathbf{Z}_q[X]/(M(X))$.]

A.1.8 Balancing CVP in Modular Lattices

Let $(L,\boldsymbol{a})$ be a closest vector problem in a modular lattice L and let $\boldsymbol{v}$ ε L be a solution. Write $\boldsymbol{a}$ as $\boldsymbol{a} = [\boldsymbol{a}_1,\boldsymbol{a}_2]$, so $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ each have N coordinates, and similarly write $\boldsymbol{v}$ as $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$. If the balancing constant b (see A.1.8) in the matrix of L is replaced by a new balancing constant b_{new} to form a new modular lattice L_{new}, then the closest vector problem $(L_{\text{new}},\boldsymbol{a}_{\text{new}})$ has the solution $\boldsymbol{v}_{\text{new}}$, where $\boldsymbol{a}_{\text{new}} = [(b_{\text{new}}/b)\boldsymbol{a}_1,\boldsymbol{a}_2]$ and $\boldsymbol{v}_{\text{new}} = [(b_{\text{new}}/b)\boldsymbol{v}_1,\boldsymbol{v}_2]$. (More precisely, $\boldsymbol{v}_{\text{new}}$ will be very close to $\boldsymbol{a}_{\text{new}}$ and the Gaussian heuristic can be used to verify that it is likely to be a closest vector.) Thus for any given modular lattice closest vector problem $(L,\boldsymbol{a})$, one solves the problem by choosing a balancing constant b and modified lattice and vector $\boldsymbol{a}$ that make the problem easiest,

In practice, it is easiest to solve a modular lattice closest vector problem $(L,\boldsymbol{a})$ if the two halves of the problem have approximately equal length. A ML CVP is said to be *balanced* if a solution $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$ ε L to the CVP satisfies

$$\| \boldsymbol{v}_1 - \boldsymbol{a}_1 \| \approx \| \boldsymbol{v}_2 - \boldsymbol{a}_2 \|.$$

It is often possible to use general knowledge about the form of the solution vector $\boldsymbol{v}$ to determine a balancing constant that makes the problem balanced. (For example, one might know that $\boldsymbol{v}_1$ is a binary vector with d_1 ones and that $\boldsymbol{v}_2$ is a binary vector with d_2 ones.) Thus in analyzing the difficulty of solving the CVP, it is advisable to always assume that the attacker knows how to balance the problem.

An equivalent definition of a balanced closest vector problem says that among all choices of balancing constant b, the ratio of the target distance $\|\boldsymbol{v} - \boldsymbol{a}\|$ to the root-discriminant $\mathrm{Disc}(L)^{1/\dim(L)} = (bq)^{1/2}$ is minimized. Thus in order to balance a closest vector problem, it is only necessary to know (approximately) the distance from a closest vector to $\boldsymbol{a}$. It is not necessary to actually know a closest vector.

A.1.9 Fundamental CVP Ratios in Modular Lattices

If the lattice L were to have a basis consisting of n equal length, pairwise orthogonal vectors, then those n basis vectors would each have length equal to the root-discriminant $\mathrm{Disc}(L)^{1/\dim(L)}$. Lattices that have such a basis are particularly easy to work with. For a closest vector problem $(L,\boldsymbol{a})$ in which the target vector is quite close to $\boldsymbol{a}$ (i.e., closer than predicted by the Gaussian heuristic), the ratio of the root-discriminant to the target distance is one measure of the difficulty of solving the problem. This ratio is denoted by

$$\rho = \rho\,(L,\boldsymbol{a}) = \lambda(L,\boldsymbol{a})/\mathrm{Disc}(L)^{1/\dim(L)}\ .$$

In general, the smaller the value of $\rho\,(L,\boldsymbol{a})$, the easier it will be to find a closest vector to $\boldsymbol{a}$. This is true because a small value of ρ means that the target vector $\boldsymbol{v}$ is probably much closer to $\boldsymbol{a}$ than it is to any other vector in L, so a lattice search algorithm will have an easier time distinguishing $\boldsymbol{v}$ from the other vectors in L.

Experimentally [B36], we observe that a more useful quantity to hold constant as the dimension increases is not σ, but the related quantity

$$c = \rho * \surd(2N).$$

Let L be a modular lattice L of dimension $n = 2N$ and modulus q. A second quantity that affects the difficulty of solving a closest vector problem in L is the ratio of the dimension to the modulus. This ratio is denoted by

$$a = \mathrm{a(L)} = N/q.$$

Experiments have suggested that holding a constant and increasing c increases lattice breaking times considerably, and that holding c constant and increasing a decreases lattice breaking times very slightly.

A.1.10 Creating a Balanced CVP for Modular Lattices Containing a Short Vector

A typical problem of cryptographic interest is to find a short target vector $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$ in a given modular lattice L of dimension $2N$, modulus q, and balancing constant $b = 1$. Assuming that $\boldsymbol{v}$ is actually a shortest vector in L, it can be found by solving the SVP for L, but one frequently knows some additional information about $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ that allows an easier CVP to be solved.

Write $\boldsymbol{v}_1 = [v_{11},v_{12},\ldots,v_{1N}]$ and $\boldsymbol{v}_2 = [v_{21},v_{22},\ldots,v_{2N}]$. In many situations one knows (or can approximate) the quantities

$$\gamma_1 = v_{11} + v_{12} + \ldots + v_{1N}\,, \qquad \delta_1 = v_{11}^{\;2} + v_{12}^{\;2} + \ldots + v_{1N}^{\;2},$$

$$\gamma_2 = v_{21} + v_{22} + \ldots + v_{2N}\,, \qquad \delta_2 = v_{21}^{\;2} + v_{22}^{\;2} + \ldots + v_{2N}^{\;2}.$$

Example. If $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are binary vectors with a specified number of zeros and ones, then it is easy to compute $\gamma_1,\delta_1,\gamma_2,\delta_2$.] The length $\|\boldsymbol{v}\|$ is larger than the distance from $\boldsymbol{v}$ to the known vector

$$\boldsymbol{d} = [d_1,\boldsymbol{a}_2] = [\gamma_1/\mathrm{N}, \gamma_1/\mathrm{N}, \ldots,\gamma_1/\mathrm{N}, \gamma_2/\mathrm{N}, \gamma_2/\mathrm{N}, \ldots,\gamma_2/\mathrm{N}],$$

so it will generally be easier to find $\boldsymbol{v}$ by solving the CVP for $(L,\boldsymbol{d})$ than it will be by solving SVP for L. The precise formulas for the relevant distances are

$$\|\boldsymbol{v}\|^2 = \delta_1 + \delta_2 \quad \text{and} \quad \|\boldsymbol{v} - \boldsymbol{d}\|^2 = \delta_1 - \gamma_1^{\,2}/\mathrm{N} + \delta_2 - \gamma_2^{\,2}/\mathrm{N}.$$

In order to balance the problem, one uses the balancing constant $b = \|\boldsymbol{v}_2 - \boldsymbol{d}_2\|/\|\boldsymbol{v}_1 - \boldsymbol{d}_1\|$ for L. Then the closest vector to $[b\boldsymbol{d}_1,\boldsymbol{d}_2]$ will probably be the vector $[b\boldsymbol{v}_1,\boldsymbol{v}_2]$. The ρ parameter for this balanced CVP is

$$\rho = [2(\delta_1 - \gamma_1^{\,2}/\mathrm{N})^{1/2}(\delta_2 - \gamma_2^{\,2}/\mathrm{N})^{1/2}/q]^{1/2}.$$

The Gaussian heuristic predicts that the balanced CVP will have a unique solution (up to obvious symmetries of the lattice) provided that the value of ρ is significantly smaller than $(N/2\pi e)^{1/2}$, which implies that the value of c is significantly smaller than $\mathrm{N}/\sqrt{(\pi e)}$.

A.1.11 Modular Lattices Containing (Short) Binary Vectors

Let

$$\mathbf{B}_N(d) = \{ \text{ binary vectors of dimension } N \text{ with } d \text{ ones and } N - d \text{ zeros } \}.$$

For example, $\mathbf{B}_4(2) = \{ [0,0,1,1], [0,1,0,1], [0,1,1,0], [1,0,0,1], [1,0,1,0], [1,1,0,0] \}$. In general the set $\mathbf{B}_N(d)$ has $N!/d!(N\text{-}d)!$ elements.

Let L be a modular lattice of dimension $2N$ and modulus q and balancing constant $b = 1$, and suppose that it is known that L contains a vector $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$ with $\boldsymbol{v}_1 \in \mathbf{B}_N(d_1)$ and $\boldsymbol{v}_2 \in \mathbf{B}_N(d_2)$. Then it is known that

$$\gamma_1 = d_1, \quad \delta_1 = d_1, \quad \gamma_2 = d_2, \quad \delta_2 = d_2.$$

The best method to search for $\boldsymbol{v}$ is to solve a balanced CVP with fundamental ratios

$$\rho = (2/q)^{1/2}(d_1(1 - d_1/N)d_2(1 - d_2/N))^{1/4} \quad \text{and} \quad \mathrm{a} = N/q.$$

If $d_1 = d_2 = d$, then the CVP is already balanced and the formulas for the fundamental ratios simplify to

$$\rho = (2d(1 - d/N)/q)^{1/2} \quad \text{and} \quad \mathrm{a} = N/q.$$

A.1.12 Convolution Modular Lattices

A *Convolution* (or *Circulant*) *Modular Lattice* (CML) is a modular lattice in which the matrix h is a circulant matrix, that is, h is a matrix of the form

$$h = \begin{bmatrix} h_0 & h_1 & \cdots & h_{N-1} \\ h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ h_1 & h_2 & \cdots & h_0 \end{bmatrix},$$

where $h_0,\ldots,h_{N\text{-}1}$ are integers, taken without loss of generality to satisfy $|h_i| \le q/2$.

A simple way to generate a convolution modular lattice containing a short vector of a specified length is to use the convolution ring $R_q = \mathbf{Z}_q[X]/(X^N–1)$. First choose two polynomials $f(X),g(X) \in R_q$ whose vectors of coefficients are short. For example, $f(X)$ might have binary coefficients with d_1 ones and $g(X)$ might have binary coefficients with d_2 ones. Then find a solution $h(X) \in R_q$ to the equation $f(X)*h(X) = g(X)$. A solution will generally exist provided $\gcd(h(1),q) = 1$; and if a solution exists, it is easily computed using the Euclidean algorithm and (if q is composite) the Chinese Remainder Theorem and Hensel's Lemma. If the coefficients of $h(X) = h_0+h_1X+h_2X^2+\ldots+h_{N-1}X^{N-1}$ are used as the upper righthand quadrant of a convolution modular lattice L_h, then the lattice L_h contains the vector

$$[\ f_0, f_1, f_2, \ldots, f_{N–1}, g_0, g_1, g_2, \ldots, g_{N–1}\ ] \ \in\ \mathbf{B}_N(d_1) \times \mathbf{B}_N(d_2).$$

The cyclic nature of a convolution lattice L means that for every vector

$$\boldsymbol{v} = [\ a_0, a_1, a_2, \ldots, a_{N-1}, b_0, b_1, b_2, \ldots, b_{N-1}\ ] \in L,$$

all of the vectors obtained by cyclically shifting the two halves of $\boldsymbol{v}$ are in L. In other words, the vectors

$$[\ a_k, a_{k+1}, a_{k+2}, \ldots, a_{k–1}, b_k, b_{k+1}, b_{k+2}, \ldots, b_{k–1}\ ],\quad k = 1, 2, 3, \ldots, N–1,$$

are also in L.

A.1.13 Heuristic Solution Time for CVP in Modular Lattices

Let L be a modular lattice of dimension $n = 2N$ and modulus q, and let $(L,\boldsymbol{v})$ be a balanced closest vector problem. Then experimental evidence [B36] [B44] suggests that the average time T to solve the closest vector problem $(L,\boldsymbol{a})$ is exponential in the dimension, with constants depending on the quantities $\mathrm{c}(L,\boldsymbol{a})$ and $\mathrm{a}(L)$ introduced in A.1.9. In other words,

$$\log(T) \approx \alpha\, N + \beta,$$

where $\alpha = \alpha(c, a)$ and $\beta = \beta(c, a)$ depend on $c = c(L, \boldsymbol{v})$ and $\boldsymbol{a} = \boldsymbol{a}\,(L)$.

This heuristic allows experimental determination of the constants α and β for given values of c and $\boldsymbol{a}$. After α and β are determined, then the formula $\log(T) \approx \alpha\, N + \beta$ can be used to extrapolate the time needed to solve a balanced closest vector problem $(L^*,\boldsymbol{v}^*)$ whose dimension $2N^*$ is too large to solve directly. Thus the following steps can be used to estimate the time to solve a modular lattice CVP:

a) Replace $(L^*,\boldsymbol{a}^*)$ by an associated balanced CVP if it is not already balanced.

b) Compute the c and a constants $c^* = c(L^*,\boldsymbol{v}^*)$ and $\mu^* = \mu(L^*)$ for the given CVP.

c) Perform experiments to solve many balanced ML CVPs $(L,\boldsymbol{v})$ whose c and $\boldsymbol{a}$ constants satisfy $c(L,\boldsymbol{a}) = c^*$ and $\boldsymbol{a}(L) = \boldsymbol{a}^*$. Do this for many different problems in each of many different dimensions $2N_i$, $i = 1,2,3,\ldots$. Record the average time T_i to solve the problems in each dimension.

d) Plot the points $(N_i,\log(T_i))$, $i = 1,2,3,\ldots$, and compute the regression line $Y = \alpha\, X + \beta$.

e) Extrapolate the solution time T^* for the original problem by the formula $\log(T^*) \approx \alpha\, N^* + \beta$.

A.1.14 Zero-forcing

If f or g have a large number of zero entries, then the zero-forcing algorithms of May and Silverman [B72] [B73] for modular convolution lattices may allow reduction of the lattice dimension. In the case that f has d1s and N-d 0s, the speedup in performing an r-fold zero-force is approximately

$$\left(1-\left(1-\prod_{i=0}^{d-1}\left(1-\tfrac{r}{N-i}\right)\right)^{N}\right)2^{\alpha r/2}$$

where the running time for the given class of lattices is $T \approx 2^{\alpha N + \beta}$. The optimal value of r may be determined using this formula. If g has more 0s than f, an attacker may invert h mod q and attempt zero-forcing in the lattice defined by h^{-1} to recover (g, f). For all the parameter sets in this standard, f has more 0s than g, so this approach will not advantage the attacker.

A.2 Experimental Solution Times for NTRU lattices – full key recovery

A.2.1 Experimental Solution Times for NTRU lattices using BKZ reduction

A private key consists of a pair of ($f(X)$,$g(X)$). The associated LBP-PKE public key $h(X)$ is formed via the relation

$$f(X) * h(X) \equiv p * g(X) \pmod q$$

The associated CML CVP formed from the coefficients of $h(X)/p$ mod q has target vector $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$ formed from the coefficients of $[f(X),g(X)]$. The selection of $f(X)$ and $g(X)$ should follow the guidelines described in this Annex for the selection of target vectors for ML CVPs. In the case that $f(X)$ has the form $f_0(X) + p*F(X)$ for a known polynomial $f_0(X)$ (e.g., $f_0(X) = 1$), then the CML CVP target vector is the vector $[F(X),g(X)]$. The security must be computed using the smaller norm bound associated to $[F(X),g(X)]$.

The CML formed using the coefficients of the public key $h(X)$ may also be used to formulate a CVP in which the target vector $\boldsymbol{v} = [\boldsymbol{v}_1,\boldsymbol{v}_2]$ is formed from the coefficients of $[r(X),m(X)]$. This lattice problem can also be described in terms of the values a and c. For the parameter sets given in this standard, the message-recovery lattice problem is slightly easier than the key-recovery lattice problem.

Table 1gives the relationship between N and lattice security levels in bits as determined experimentally for convolution modular lattices. Experiments were run using Victor Shoup's NTL library [B95]. Lattices with the given values of c and a were successfully reduced at low dimension, and the figures given below were obtained by a least-squares fit to the points corresponding to the values of N that required more than 35 bits of effort to reduce (this value varied depending on c and a). It was observed that holding a constant and increasing c increased lattice breaking times considerably, and that holding c constant and increasing a decreased lattice breaking times very slightly. Here,

$$c = \sqrt{(4e\ \|F\|\ \|g\| / q)}.$$

The experiments were run on 400 MHz Celeron machines, and the time in seconds converted to the time in MIPS-years by first multiplying by 400 (to account for the 400 MHz machines) and then dividing by 31557600, which is the number of seconds in a year. Breaking times were converted to bit security using the identification of 80-bit security with 10^{12} MIPS-years [B70].

Table 1 – Lattice Security

c	a	Bit Security
1.73	0.53	0.3563N - 2.263
2.6	0.8	0.4245N - 3.440
3.7	2.7	0.4512N + 0.218
5.3	1.4	0.6492N - 5.436

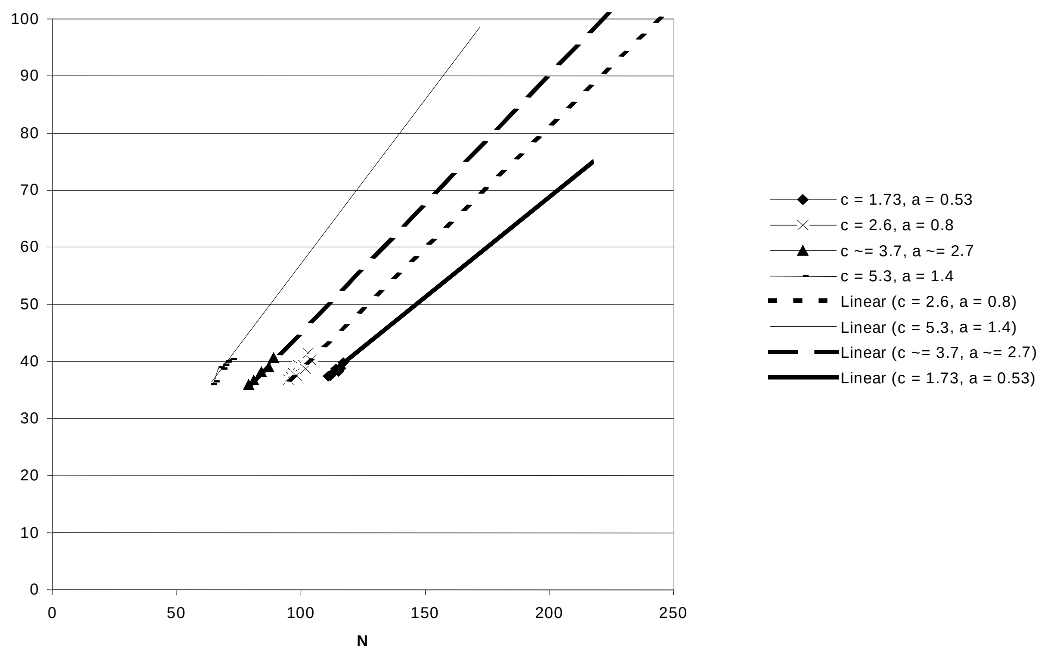

Figure 4: Lattice Breaking Times and Linear Extrapolations

There is some variation among published estimates of running time due to the particular definition of a MIPS-Year and to the difficulty of estimating actual processor utilization. (How many arithmetic instructions a modern processor performs in a second when running an actual piece of code depends heavily not only on the clock rate, but also on the processor architecture, the amount and speeds of caches and RAM, and the particular piece of code.) Thus, the estimates given here may differ from others in the literature, although the relative order of growth remains the same. We note that the current estimates of lattice strength allow a large margin for error and for improvements in lattice reduction techniques.

NOTE—The strength of any cryptographic algorithm relies on the best methods that are known to solve the hard mathematical problem that the cryptographic algorithm is based upon. The discovery and analysis of the best methods for any hard mathematical problem is a continuing research topic. Users of LBP-PKE should monitor the state of the art in lattice reduction, as it is subject to change.

A.2.2 Alternative Target Vectors

Examination of the NTRU decryption process reveals that any sufficiently small (f', g'), with the property that $f' \cdot h = p \cdot g' \bmod q$, will allow decryption. [B19] observes that, with slightly longer vectors, it might be possible to decrypt with sufficient accuracy to allow an attacker to complete the decryption by brute force. Neither of these attacks appears to be feasible. Although NTRUSign [B32] makes use of the existence of short vectors that are linearly independent of f and g, it has been observed experimentally [B30][B36] that lattice reduction techniques that find any vector shorter than q will in fact terminate with (f, g) or one of its trivial “rotations” ($f \cdot X^k$, $g \cdot X^k$). Thus, there is not currently known to be an attacker who can mount an attack based on slightly longer short vectors but does not know the short vectors themselves.

A.3 Combined Lattice and Combinatorial Attacks on LBP-PKE Keys and Messages

A.3.1 Overview

[B39] presents a method for combining lattice reduction and combinatorial attacks. We refer to this attack as a “hybrid” attack. In this approach, an attacker performs a certain amount of work to reduce the central part of an NTRU lattice. Following the reduction, rows 1 to y_1 -1, $y_1 < N$, are unreduced, rows y_1 to y2, $N < y_2 < 2N$, are reduced, and rows y_2+1 to 2N are unreduced. Let $K = 2N-y_2$ be that part of the lattice containing the private key *f* that remains unreduced. The attacker can perform a combinatorial search for the part of the key containined in the K-dimensional subspace. The attacker guesses the coefficients of the part of *f* in this subspace and sums the lower K rows of the lattice to construct a 2N-dimensional vector.If the guess is correct, the first y2 entries in the vector will be very close to a point in the y_2-dimensional transformed lattice that was output by the original reduction process.

The attack thus has two stages: the lattice reduction stage and the combinatorial stage. The total time for the attack is the sum of the time for these stages. This standard requires that for a security level *k*, both of these stages shall take more than *k* bits of work.

A.3.2 Lattice Strength

In a hybrid attack, the lattice is not completely reduced. Instead, the attacker selects a sublattice of the main lattice and applies a lattice reduction algorithm to that sublattice. This sublattice will, with high probability, not include any vector with length shorter than the Gaussian value discussed in A.1.5. The lattice running times given in A.2 are for full key recovery; in this case, a short vector is present, and this reduces lattice reduction times. In the hybrid case, where no short vector is present, the experiments of A.2 no longer apply and, rather than measuring the running times necessary to recover the short vector, the new experiments measure the amount of reduction that can be performed in a given amount of time. In this case, the amount of reduction is measured by the number of diagonal entries b_i in the lattice that can be altered by the reduction process so they take a value other than *q* or 1.

Figure 5 presents the results of a number of lattice experiments for $q = 2048$, also presented in [B30]. The experimental results fall into three clusters corresponding to three different experimental techniques: standard BKZ, given by the points in the bottom left corner; the isodual technique described in [B39], given by the points in the top half of the graph around the middle; and a refinement of the isodual technique in which the output from each blocksize (where blocksize is a fundamental tuning parameter) is used as the input into the next blocksize rather than running each blocksize on the original, unreduced lattice [B30]. As demonstrated by Figure 5, this final technique is the best one known to date.

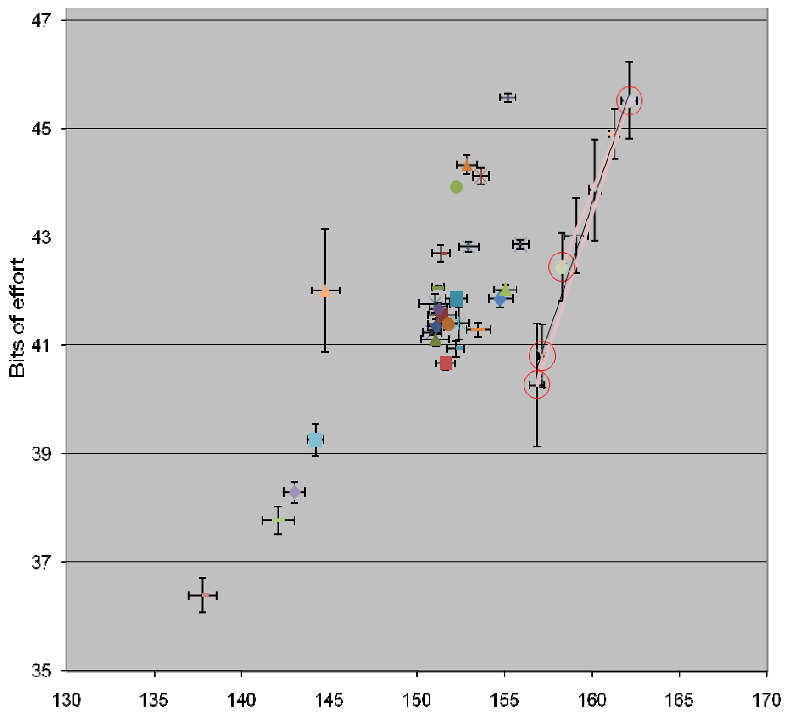

Figure 5: Time to remove *x* q vectors by different lattice reduction techniques, experimentally determined.

Based on this data, it appears the running time t to remove a given number N_q of q-vectors using the best currently known method is given by

$$t = 0.9501N_q - 3 \ln (2 N_q) - 123.58$$

A.3.3 Reduced lattices and the "cliff"

A.3.3.1 Running time to obtain a given profile

An attacker's chance of successfully recovering the private key depends on the values on the diagonal entries of the reduced lattice. We refer to the set of the logs of these values as the lattice's "profile". For convenience we take logs base q, so a profile goes from 1 to 0. Figure 6 presents a set of reduced profiles. If a profile does not go continuously to 0, we say it has a "cliff" of height α.

The running time to obtain a slope δ if there is no cliff can be related straightforwardly to the time to remove N_q q-vectors: if there is no cliff, the reduction is symmetric about N (in order to keep the determinant constant) so the slope $\delta = 1/(y_2 - y_1) = 1/2N_q$.

The time to obtain a cliff of height α, occurring at location $N < y_2 < 2N$ in the profile, is related to the time to obtain a slope δ with no cliff as follows [B30]: if

$$\log_2(t) = m/\delta + 3\ln(1/\delta) + c \text{, where in this case } t = 0.4750/\delta + 3\ln(1/\delta) - 123.58,$$

then

$$\log_2(t) = 2m\frac{(y_2 - N)}{(1-\alpha)^2} + 3\ln(y_2 - y_1) + c \ .$$

Since lattice attacks are continually improving, the parameter sets in this standard are generated by assuming the following extrapolation line:

$$t = 0.2/\ \delta - 3 \ln (1/\ \delta) - 50.$$

This grants the attacker considerably more power than they are currently known to have.

A.3.3.2 The cliff height α and p_s

For a given amount of work, the attacker may choose from a range of (y_2, α) pairs.

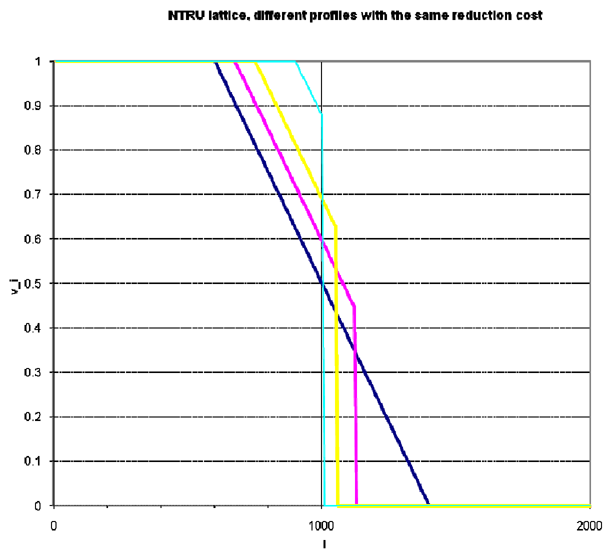

Figure 6 Lattice Profiles

Having performed the reduction, the attacker has the view of the lattice shown in Figure 7. The middle section of the lattice contains some rotation of a part of g and a part of f. The attacker will mount an attack consisting of an enumeration through the substring of f in the unreduced part of the lattice on the right, combined with reduction against the reduced part of the lattice in the middle and the unreduced part on the left. The enumeration of the substring of f is speeded up using meet-in-the-middle techniques.

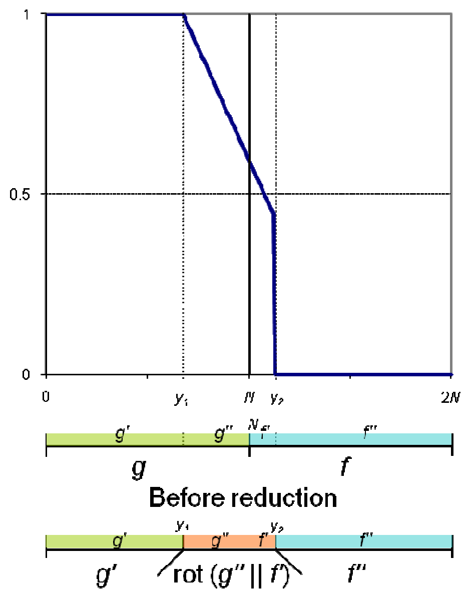

Figure 7 The attacker's view of the lattice following reduction

If the attacker has correctly guessed *f*' and *f*'' such that *f*' + *f*'' makes up the part of the key *f* that lies in the unreduced section $y_2 < i < 2N$, they can confirm this guess by reducing against the rest of the lattice, $0 < i < y_2$. The most efficient way of carrying out this reduction is by using Babai's method [B9], which has a running time of about N^2. However, this reduction method has a chance of failing: if any term in the part of the key that lies in the reduced area is greater than the corresponding diagonal term, the Babai reduction will not be successful. Figure 8 gives an example where the Babai reduction will fail. This illustrates that if there is a "cliff" in the profile, the Babai reduction is much more likely to succeed.

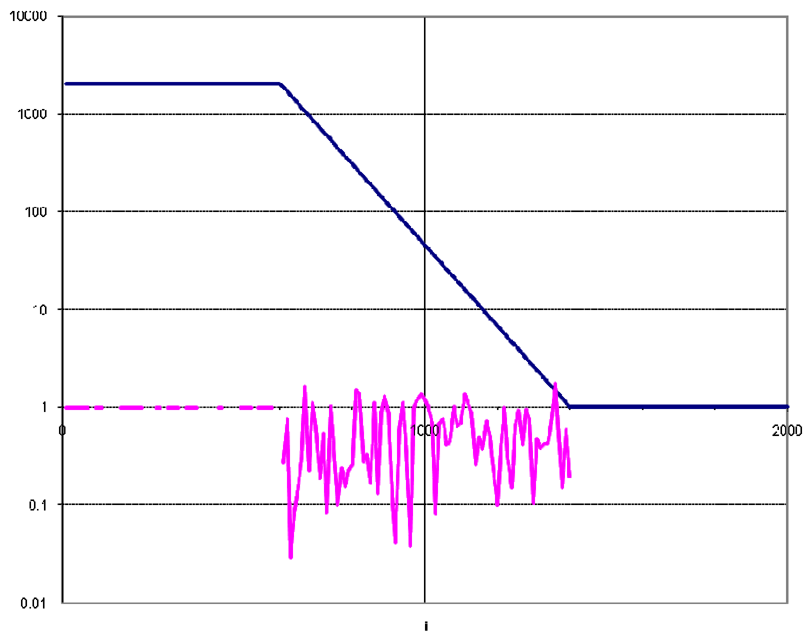

Figure 8: A case where Babai reduction will not be successful

The probability of success at this stage, given an *f'* and and *f''* that should make *f*, is denoted by p_s. This value p_s depends on *N*, *q*, the height of the cliff α, and the boundaries of the reduced area (y_1, y_2), and is given by [B30]:

$$\begin{aligned} p_s &= \left(1-\frac{2}{3q}\right)^{y_1} \prod_{i=0}^{y_2-y_1} \left(1-\overline{f_{q^{\frac{\alpha(y_2-y_1)+i(1-\alpha)}{y_2-y_1}},\sigma}}\right) \\ &= \left(1-\frac{2}{3q}\right)^{\frac{2N-y_2(1+\alpha)}{1-\alpha}} \prod_{i=0}^{\frac{2(y_2-N)}{1-\alpha}} \left(1-\overline{f_{q^{\frac{2\alpha(y_2-N)+i(1-\alpha)^2}{2(y_2-N)}},\sigma}}\right) \end{aligned} ,$$

where

$$\overline{f_{D,\sigma}} = \operatorname{erfc}\left(\frac{D}{\sigma\sqrt{2}}\right) - \frac{\sigma\sqrt{2}}{D\sqrt{\pi}}\left(e^{-\frac{D^2}{2\sigma^2}}-1\right).$$

A.3.4 Combinatorial Strength

This section considers the effort that the attacker must expend in the combinatorial phase of the combined attack.

A.3.4.1 Combinatorial Attacks on LBP-PKE Keys and Messages

An exhaustive search algorithm tries all allowable values for $\boldsymbol{v}_1$, computes the value of $\boldsymbol{v}_2 = \boldsymbol{v}_1 * h$, and checks if $\boldsymbol{v}_2$ is an allowable value. Let S_1 denote the sample space for $\boldsymbol{v}_1$. The exhaustive search method has average running time ½$|S_1|$ for general modular lattices and average running time (1/2N)$|S_1|$ for convolution modular lattices (since a convolution modular lattice will generally have *N* target vectors). An exhaustive search algorithm has no storage requirements.

A collision search algorithm of Odlyzko is described in [B43][B44].

If $S_1 = \mathbf{B}_N(d)$ is the space of binary vectors of dimension *N* with *d* ones, then the running time of the collision search method is approximately $d^{1/2}\mathrm{C}(N/2,d/2)$ operations. (Here $\mathrm{C}(n,k) = n!/k!(n–k)!$ is the usual combinatorial symbol.) The storage requirement is approximately $2\mathrm{C}(N/2,d/2)$.

If $S_1 = \mathbf{T}_N(d)$ is the space of trinary vectors of dimension *N* with *d* 1s and *d* -1s, then the running time of the collision search method is approximately $d^{1/2}\mathrm{C}(N,d)$ operations. The storage requirement is approximately $2\mathrm{C}(N,d)$.

It is not known if there is a collision search method that does not require substantial storage, but it is recommended that security be computed under the assumption that storage requirements are not an issue (a contrary view is given in [B99]).

A.3.4.2 Combinatorial Strength in the hybrid case

In the hybrid case the attacker is searching a space of size K for a trinary polynomial with c_1 +1s and c_2 -1s. The amount of work the attacker must do to search this space using a standard collision search method is:

$$W_{search} = \frac{\binom{K}{c_1/2}\binom{K-c_1/2}{c_2/2}}{\sqrt{\binom{c_1}{c_1/2}\binom{c_2}{c_2/2}}}.$$

Wagner's generalized birthday paradox search [B102] presents an attack that may potentially improve the running time of this stage to

$$W_{search} = \frac{\binom{K}{c_1/2}\binom{K-c_1/2}{c_2/2}}{\binom{c_1}{c_1/2}\binom{c_2}{c_2/2}}.$$

It is not clear exactly how this attack would be implemented against the current form of LBP-PKE. Nevertheless, the parameter sets presented in this standard for a given security level *k* assume the attacker can mount this generalized birthday paradox attak and so use the second form for W_{search}.

W_{search} contributes to the full security level of the combinatorial search phase. Two additional contributions to this security level are: first, the chance that the search is not successful; second, the cost of performing the reduction against the rest of the lattice.

The chance that the search is not successful depends on two quantities:

The chance that the lattice reduction allows a correct guess to be confirmed, p_s. The value for p_s is given above. For the standard attack, the search work becomes W_{search} / √p_s. For the generalized attack, the search work becomes W_{search}/p_s. We express the total search work as $W_{search}*W_{p\_s}$.

The chance that the attacker has guessed the right values for c1, c2, $P_{split}(c_1, c_2; N, K, d_1, d_2)$. Here the analysis is complicated by the fact that the lattice in fact contains *N* rotations of the private key. The chance that the attacker has guessed the right values for c1 and c2 for a single rotation of the key is

$$P_{split,1} = \frac{\binom{N-K}{d_1-c_1}\binom{N-K-(d_1-c_1)}{d_2-c_2}\cdot\binom{K}{c_1}\binom{K-c_1}{c_2}}{\binom{N}{c_1}\binom{N-c_1}{c_2}}$$

If the attacker can take advantage of the fact that the lattice contains N rotations of the key, P_{split} improves to become

$$P_{split,N} = 1 - (1 - P_{split,1})^N.$$

It is currently believed that the form of the private key, $f = 1 = pF$, requires the attacker to solve a CVP problem that "locks in" a single rotation of the key, and so the appropriate measure of P_{split} is $P_{split,\ 1}$. However, for safety against an improved reduction algorithm that would let an attacker search against all rotations of the key, the parameters in this standard were generated with $P_{split} = P_{split,\ N}$.

Finally, in the specific setting of the hybrid attack, the reduction using Babai's method involves multiplying by a 2Nx2N transformation matrix; experimentally it is found that this multiplication has bit security about

$$W_{reduction} = N^2/2^{1.06}.$$

Since the matrix is the same in all cases, this security level can probably be optimized, and for purposes of estimating security it is taken to have the value

$$W_{\text{reduction}} = N/2^{1.06} .$$

This time must be multiplied by the search time of the meet-in-the-middle part of the attack to obtain the full running time of this phase of the hybrid attack.

The total expected work of this phase for a given choice of c_1, c_2, given the values K, α, y_1, and y_2 that resulted from the lattice reduction phase, is therefore

$$W_{\text{mitm}} (c_1, c_2) = W_{\text{reduction}} * W_{\text{search}} * W_{\text{p\_s}} / P_{\text{split}}.$$

Finally, the security level due to this phase is taken to be

$$W_{\text{mitm}} = \min_{c1,\, c2} W_{\text{mitm}} (c_1, c_2).$$

A.3.5 Summary

A hybrid attack involves the lattice reduction work, W_{latt}, and the meet-in-the-middle work, W_{mitm}. The attacker will attempt to balance these two phases so that they take equal amounts of time. A parameter set has a strength of greater than *k* bits if, for all profiles produced by performing *k* bits of lattice reduction, the value of $W_{\text{mitm}} > k$.

A.4 Other Security Considerations for LBP-PKE Encryption

A.4.1 Entropy Requirements for Key and Salt Generation

The security of a parameter set will be less than the claimed level if an attacker can guess either the key or the random padding with less effort than a brute-force search. One means of doing this would be for the attacker to guess the internal state of the random number generator used in key generation and salt generation. These RNGs must be seeded with the appropriate amount of entropy, which is *k*+64 for a claimed security level *k*.

A.4.2 Reduction mod *q*

If the calculation of *rh* mod q involves little or no reduction mod *q*, an attacker can attempt to use their knowledge of *h* to solve *e* = *rh* + *m*’ using linear algebra. For the parameter sets in this standard, it is vanishingly unlikely that this will occur if *h* is a valid public key. The public key partial validation method given in 9.2.5.2.2 checks that it is highly likely that the calculation of *r***h* will involve significant reduction mod *q*.

A.4.3 Selection of *N*

It is required that the security parameter *N* be prime (i.e., the dimension *n* of the lattice be twice a prime). If *N* is highly composite (e.g., if *N* is a power of 2), then Gentry [B23] has shown that a folding method allows the private key and plaintext to be recovered from a lattice of dimension much smaller than *N*.

A.4.4 Relationship between *q* and *N*

It is recommended that for each prime divisor q_0 of *q*, the polynomial $X^N - 1$ modulo q_0 should have no factors of small degree (aside from the obvious factor $X - 1$). If *N* is prime, then $X^N - 1$ modulo q_0 factors as $(X - 1)A_1(X)\ldots A_e(X)$, where each $A_i(X)$ has degree equal to the multiplicative order of q_0 modulo *N*. If $h(X)$ or $r(X)$ is zero in the field mod $A_i(X)$, it will leak the value of m'(X) in this field. If $A_i(X)$ has degree *t*, the probability that $h(X)$ or $r(X)$ is divisible by $A_i(X)$ is presumably $1=q^t$. To avoid attacks based on the factorization of *h* or *r*, we will require that for each prime divisor *P* of *q*, the order of *P* (mod *N*) must be N-1 or (N-1)/2. This requirement has the useful side-effect of increasing the probability that a randomly chosen *f* will be invertible in R_q.

A.4.5 Form of *q*

So long as the factors of *q* have sufficient order mod *N* (A.4.5), there are no known security issues with the form of *q*: it may be chosen to be either prime or composite. This standard selects *q* to be 2^l for some *l* to increase the efficiency of the modular reduction operations.

A.4.6 Leakage of *m*'(1)

Because X^N-1 is always divisible by *X*-1, the mapping f(*X*) → f(1) is a *ring homomorphism*, i.e

$$(f*g)(1) = f(1)g(1).$$

Note that *f*(1) is simply the sum of the coefficients of *f*. Since an attacker will be able to calculate *h*(1), and since *r*(1) is part of the parameter set, this means that an attacker can recover *m*'(1) from *e*=*r***h*+*m*'. The attacker could potentially distinguish between two *m*'s by their Hamming weight. This is addressed by the masking process, which ensures that *m*'(1) does not leak information about *m*(1); see A.4.8 for further details.

For binary keys, m'(1) reveals the number of 1s in m'. Since lattice and combinatorial attacks on (*r*, *m*') get easier as *m*' gets more unbalanced (in other words, as the number of 1s gets further and further from *N*/2), an attacker can select (*r*, *m*') pairs that are more vulnerable to these attacks based on the revealed value of *m*'(1). However, for trinary keys and messages (including product-form trinary keys), m'(1) is simply the number of 1s minus the number of -1s and does not directly reveal information about more versus less vulnerable message representatives.

A.4.7 Relationship between *p*, *q* and *N*

If the smaller modulus *p* divides the large modulus *q*, then reduction modulo *p* of an expression *p***r***h* + *m* modulo *q* will immediately recover *m*. More generally, if *p* and *q* are not relatively prime in the ring $\mathbf{Z}[X]/(X^N - 1)$, then reduction modulo a common factor will reveal information about *m*. For this reason it is required that the large modulus *q* and the smaller modulus *p* be relatively prime in the ring $\mathbf{Z}[X]/(X^N - 1)$. This is equivalent to the condition that the three quantities *q*, *p*, and $X^N - 1$ must generate the unit ideal in the ring $\mathbf{Z}[X]$.

The large modulus *q* is required to be in **Z**, but the smaller modulus *p* need not be in **Z**. For example, if *N* is odd and if *q* is a power of 2, then *p* could equal $X + 2$ or $X - 2$, since the three quantities $X^N - 1$, 2^k, and $X \pm 2$ generate the unit ideal in the ring $\mathbf{Z}[X]$.

A.4.8 Adaptive Chosen Ciphertext Attacks

If the same r is used to encrypt two different message representatives m'_1 and m'_2 under the same key, then the difference of the two ciphertexts $e_1 - e_2 \equiv m'_1 - m'_2$ (mod q) will reveal a large portion of m'_1 and m'_2. With the encryption schemes in this standard, $m' = M \oplus \text{MGF}(r*h) = M + \text{MGF}(r*h) \bmod 2$, so $e_1 - e_2$ (mod q)(mod 2) = $M_1 \oplus M_2$. With the key establishment schemes in this standard, there are two ways that an r could be repeated:

a) The same message m could be encrypted twice with the same salt b.

b) Two different (m, b) pairs could produce the same r.

If the same message m is encrypted twice with the same salt b, an attacker will know that this has happened but will not obtain any additional information about m or b. Since this standard is a key establishment standard and the m should therefore be chosen at random for each message sent, the chance that an (m, b) pair will be used twice should be the chance of a collision in the entire (m, b) space, which requires the sending of about $2^{N/2}$ messages.

The chance that two different (m, b) pairs will produce the same r is the chance of a collision when selecting from the space of all possible blinding polynomials, D_r. In order to have a significant chance of a collision, the attacker must observe about $\sqrt{(\# D_r)}$ messages, or $\sqrt{(C(N,d)/N)}$, where C is the usual combinatorial symbol. For the parameter sets in this document, this number of messages is always greater than the number of operations an attacker must perform to mount a combinatorial attack against a key or ciphertext (see A.3.4.1).

A single message element $m(X)$ should not be encrypted using two different blinding elements. If $m(X)$ is encrypted using $r_1(X)$ and $r_2(X)$, then the quantity

$$(ph(X))^{-1}(e_1(X) - e_2(X)) \equiv r_1(X) - r_2(X) \pmod q$$

will reveal a large portion of $r_1(X)$ and $r_2(X)$. (Even if $h(X)^{-1} \bmod q$ does not exist, one may still gain considerable information using a partial inverse).

In general, as with all public-key cryptosystems, the LBP-PKE primitives must be within an appropriate encryption scheme to provide security against chosen plaintext, chosen ciphertext and adaptive chosen ciphertext attacks [B37] [B45] [B57] [B81]. The scheme used in this standard has a proof of security in the random oracle model presented in [B45]. In this model, the salt b that is added to the message before encryption is not vulnerable to birthday paradox-type attacks, but only to exhaustive search-type attacks. For a k-bit security level, it is therefore appropriate to take the salt length db to be k bits.

A.4.9 Invertibility of *g* in R_q

The proof of security in [B45] requires h, and therefore g, to be invertible in R_q. This is the reason for the check in step j) of the key generation operation in 9.2.1. There are no specific known attacks that apply only if g is not invertible. Note that, even if h is not invertible, there will often be a "pseudo-inverse" which plays the same role [B81]; this is not taken into account in the proof in [B45].

A.4.10 Decryption Failures

On decryption, the decryptor calculates

$$a = f*e \bmod q$$

$$= prg + m' + pFm' \bmod q$$

Decryption depends on this equality holding over the integers, not simply mod q. Presentations of LBP-PKE in other fora in the past have used parameter sets for which the value of q or the mod q reduction method would not always make it possible to satisfy this equality. Therefore, decryption would occasionally fail. An attacker who observed decryption failures could recover the private key [B41] [B57] [B74] [B85] [B97] even if the underlying encryption scheme was CCA2-secure in the absence of decryption failures.

For trinary polynomials with d +1s and the same number of -1s, the chance of a decryption failure is given by [B30]:

$$\mathrm{Prob}_{(q,\, d,\, N)}(\text{Decryption fails}) = P_{(d,\, N)}((q-2)/6)$$

Where

$$P_{(d,\, N)}(c) = N * \mathrm{erfc}\ (c\ /\ (\sigma\sqrt{[2N]}))$$

and

$$\sigma(d, N) = \sqrt{(8d/3N)}$$

A.4.11 OID

The OID is included in step j) of encryption and step q) of decryption to give an assurance that encrypters are using the encryption scheme specified in this document. This protects against *modified parameter attacks* [B42], in which an attacker persuades an encrypter to encrypt with an encryption scheme other than the one the decrypter specifies use with that key. Under certain circumstances, modified parameter attacks can recover information about the ciphertext. The inclusion of the OID ensures that a message will only decrypt correctly if it was encrypted with the exact parameter set expected by the receiver.

A.4.12 Use of Hash Functions by Supporting Functions

The security requirements on a hash function when used as the core of a random bit string generator are different from those on a fixed-length hash function. This standard follows common practice in using SHA-1 in Random Bit Generators at security levels up to k=128, and SHA-256 at security levels up to k=256.

A.4.13 Generating Random Numbers in [0, *N*-1]

The BPGM method (**8.3.1.1**) converts a random bit or byte stream to a series of integers. These integers must be uniformly distributed in the range [0, N-1]. If they were not, an attacker could exploit the bias to speed up an attack based on guessing r. The method given in this document ensures that the numbers are unbiased by:

— selecting a set of bits;

— converting the bits to an integer;

— only reducing the integer mod N if it falls into a range [0, kN-1] for some parameter-set-specific value k, and otherwise selecting a fresh set of r random bits.

The output of the random bit string generator must be statistically random; there should be no simple (eg linear) relationship between the sets of bits chosen for reduction.

The number of bytes chosen pre-reduction is the minimum number necessary to hold *N*. The number of bits chosen from these bytes (denoted by *c* in the parameter sets) is selected to give the minimum value of (2^c mod *N*). There are no known security implications to the choice of *c*, so long as $2^c > N$.

A.4.14 Attacks based on variation in decryption times

The paper **[B98]** demonstrates that a naïve implementation of the BPGMs in this standard (without the minimum IGF output parameter *minCallsR*) leaks private key information because the decryption time depends on the ciphertext. To prevent these attacks, it is necessary to ensure that decryption takes constant time (or at least that variations in time occur with negligible probability).

The paper **[B98]** suggests that effectively constant decryption times can be obtained by choosing *oLenMin* such that the chance that more than *oLenMin* octets of output are needed is less than 2^{-k}, where *k* is the security parameter and *oLenMin* = *minCallsR* * *hLen*, *hLen* the length in octets of output from the hash function. The chance that greater than *oLenMin* individual octets are needed is given by

$$1 - \sum_{dr<d<oLenMin/c'} P_{2^c,N,n}(oLenMin/c', d)$$

where $P_{C,N,N}(L,d)$ is determined by the recursive formula

$$P_{C,N,n}(L,d) = P_{C,N,n}(L-1,d-1)\cdot\left(\frac{n(N-d+1)}{C}\right) + P_{C,N,n}(L-1,d)\cdot\left(1-\frac{n(N-d)}{C}\right),$$

$$P_{C,N,n}(L,d) = 0 \text{ if } L < d,$$

$$P_{C,N,n}(L,0) = \left(1-\frac{nN}{C}\right)^L,$$

and

$$C = 2^c,\ c' = \text{ceil}[c/8].$$

minCallsR should be taken to be the smallest integer such that the chance that more than *oLenMin* octets of output are needed is less than 2^{-k}.

A.4.15 Choosing to attack r or m

An attacker may choose to mount an attack on a ciphertext to recover either r or i; recovering one of these trivially recovers the other. The attacker will choose to attack whichever is thinner. Since *i* is chosen at random from the space of trinary polynomials, if r is thick (as is the case for the size-optimized parameters in this standard), *i* will in general be thinner and may be easier to recover than the intended security level.

To mitigate this risk, the encryption scheme in this standard requires that an sender discards an encrypted message if the message representative *i* has fewer than *dr* +1s, -1s, or 0s. If the sender generates such a message representative, they must discard that message representative and restart the encryption process with a different salt *b*. If the receiver receives a ciphertext that decrypts to a message representative *i* with fewer than *dr* +1s, -1s, or 0s, the receiver must treat the decryption as having failed (though the receiver should complete all the stages of decryption in order to avoid leaking timing information about the cause of the decryption failure).

A.4.16 Quantum Computers

All cryptographic systems based on the problems of integer factorization, discrete log, and elliptic curve discrete log are potentially vulnerable to the development of an appropriately sized quantum computer, as algorithms for such a computer are known that can solve these problems in time polynomial in the size of the inputs. For LBP-PKE [B71], proposes a quantum lattice reduction algorithm that may improve reduction speeds while remaining exponential-time, and [B86][B100][B66][B87][B46] consider potential sub-exponential algorithms for certain lattice problems.

A.4.17 Other Considerations

The private-key representation does not affect security in general, although the effectiveness of physical attacks may vary according to the representation. The private key should be stored securely, and the encryption blinding polynomial should be erased after use. The domain parameters should be protected from unauthorized modification.

A.5 A Parameter Set Generation Algorithm

This section describes an algorithm that may be used to generate parameter sets with a desired level of security.

a) Set a desired security level *k*

b) Set q = 2048.

c) Choose a performance metric. Possible metrics include size = N* $\log_2(q)$; operation time = N*d; or some combination of the two, such as speed^2*size.

d) Set N equal to the first prime greater than k such that the order of 2 mod N is (N-1) or (N-1)/2 and enter the following loop

    1) For each d, 1 < d < N/3:

        i) For each possible N < y_2 < 2N:

            i) For each 0 < y_1 < N:

                i) Calculate the profile produced by *k* bits of lattice reduction for that $y_2\ y_1$.

                ii) If such a profile exists, calculate W_{mitm} using the formula given in A.3.4.2

                iii) If $W_{mitm} < k$, that value of d does not give sufficient security. Increment d by one and re-enter the y_2 loop.

        ii) We have now obtained the minimum value of d for the given N that gives k bits of security. Check that the value of d in question has a decryption failure probability of < 2^{-k} using the formula given in A.4.10.

        iii) If the decryption failure probability is > 2^{-k}, increase N to the next prime such that the order of 2 mod N is (N-1) or (N-1)/2 and re-enter the d loop

        iv) Return d.

    2) Calculate the “goodness” of the parameter set (N, d, q) using the chosen metric.

    3) Increase N to the next prime such that the order of 2 mod N is (N-1) or (N-1)/2 and re-enter the d loop

e) Output the stored (N, d, q) that give the best score under the chosen metric.

The parameter sets in this standard were generated to minimize running time and to minimize size. With this parameter generation algorithm it is possible to generate parameters that satisfy arbitrary performance criteria, such as “the fastest operations with a key size of less than 5000 bits”.

A.6 Possible Parameter Sets

This section defines specific sets of parameters for the encryption scheme (SVES) that give a specific level of security according to the metrics in this standard.

A.6.1 Size-Optimized

These parameter sets are optimized for size at a given security level.

A.6.1.1 ees401ep1

This parameter set is suitable for use at the 112-bit security level

Table 2 – ees401ep1
N = 401
p = 3
q = 2048
Key generation: KGP-3 with
df = 113
dg = 133
lLen = 1
db = 112
maxMsgLenBytes = 60
bufferLenBits = 600
bufferLenTrits = 400
dm0 = 113
MGF-TP-1 with
SHA-1 (MGF)
BPGM2 with
IGF-MGF-1 with SHA-1 (IGF)
dr = 113
c = 11
minCallsR = 32
minCallsMask = 9
OID = 00 02 04
pkLen = 114

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is 0.023276.

A.6.1.2 ees449ep1

This parameter set is suitable for use at the 128-bit security level

Table 3 – ees449ep1

Table 3 – ees449ep1
N = 449
p = 3
q = 2048
Key generation: KGP-3 with
df = 134
dg = 149
lLen = 1
db = 128
maxMsgLenBytes = 67
bufferLenBits = 672
bufferLenTrits = 448
dm0 = 134
MGF-TP-1 with
SHA-1 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-1 (IGF)
dr = 134
c = 9
minCallsR = 31
minCallsMask = 9
OID = 00 03 03
pkLen = 128

NOTE— If a message representative m' has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m' is 0.10411.

A.6.1.3 ees653ep1

This parameter set is suitable for use at the 192-bit security level

Table 4 – ees653ep1
N = 653
p = 3
q = 2048
Key generation: KGP-3 with
df = 194
dg = 217
lLen = 1
db = 192
maxMsgLenBytes = 97
bufferLenBits = 976
bufferLenTrits = 652
dm0 = 194
MGF-TP-1 with
SHA-256 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-256 (IGF)
dr = 194
c = 11
minCallsR = 34
minCallsMask = 9
OID = 00 05 03
pkLen = 192

NOTE— If a message representative m' has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m' is 0.043282.

A.6.1.4 ees853ep1

This parameter set is suitable for use at the 256-bit security level

Table 5 – ees853ep1
N = 853
p = 3
q = 2048
Key generation: KGP-3 with
df = 268
dg = 284
lLen = 1
db = 256
maxMsgLenBytes = 126
bufferLenBits = 1272
bufferLenTrits = 852
dm0 = 268
MGF-TP-1 with
SHA-256 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-256 (IGF)
dr = 268
c = 10
minCallsR = 42
minCallsMask = 11
OID = 00 06 03
pkLen = 256

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is 0.22669.

A.6.2 Cost-Optimized

These parameter sets are optimized to give the lowest value of (operation time)^2*size.

A.6.2.1 ees541ep1

This parameter set is suitable for use at the 112-bit security level

Table 6 – ees541ep1

Table 6 – ees541ep1
N = 541
p = 3
q = 2048
Key generation: KGP-3 with
df = 49
dg = 180
lLen = 1
db = 112
maxMsgLenBytes = 86
bufferLenBits = 808
bufferLenTrits = 540
dm0 = 49
MGF-TP-1 with
SHA-1 (MGF)
BPGM3 with
IGT-MGF-1 with SHA-1 (IGF)
dr = 49
c = 12
minCallsR = 15
minCallsMask = 11
OID = 00 02 05
pkLen = 112

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-133.39}$.

A.6.2.2 ees613ep1

This parameter set is suitable for use at the 128-bit security level

Table 7 – ees613ep1
N = 613
p = 3
q = 2048
Key generation: KGP-3 with
df = 55
dg = 204
iLen = 1
db = 128
maxMsgLenBytes = 97
bufferLenBits = 912
bufferLenTrits = 612
dm0 = 55
MGF-TP-1 with
SHA-1 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-1 (IGF)
dr = 55
c = 11
minCallsR = 16
minCallsMask = 13
OID = 00 03 04
pkLen = 128

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-151.78}$.

A.6.2.3 ees887ep1

This parameter set is suitable for use at the 192-bit security level

Table 8 – ees887ep1

N = 887
p = 3
q = 2048
Key generation: KGP-3 with
df = 81
dg = 295
iLen = 1
db = 192
maxMsgByteLen = 141
bufferLenBits = 1328
bufferLenTrits = 886
dm0 = 81
MGF-TP-1 with
SHA-256 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-256 (IGF)
dr = 81
c = 10
minCallsR = 13
minCallsMask = 12
OID = 00 05 04
pkLen = 192

NOTE— If a message representative m' has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m' is $2^{-214.25}$.

A.6.2.4 ees1171ep1

This parameter set is suitable for use at the 256-bit security level

Table 9 – ees1171ep1

Table 9 – ees1171ep1
N = 1171
p = 3
q = 2048
Key generation: KGP-3 with
df = 106
dg = 390
lLen = 1
db = 256
maxMsgLenBytes = 186
bufferLenBits = 1752
bufferLenTrits = 1170
dm0 = 106
MGF-TP-1 with
SHA-256 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-256 (IGF)
dr = 106
c = 10
minCallsR = 20
minCallsMask = 15
OID = 00 06 04
pkLen = 256

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-283.49}$.

A.6.3 Speed-Optimized

These parameter sets are optimized for speed at a given security level.

A.6.3.1 ees659ep1

This parameter set is suitable for use at the 112-bit security level

Table 10 – ees659ep1

Table 10 – ees659ep1
N = 659
p = 3
q = 2048
Key generation: KGP-3 with
df = 38
dg = 219
lLen = 1
db = 112
maxMsgLenBytes = 108
bufferLenBits = 984
bufferLenTrits = 658
dm0 = 38
MGF-TP-1 with
SHA-1 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-1 (IGF)
dr = 38
c = 11
minCallsR = 11
minCallsMask = 14
OID = 00 02 06
pkLen = 112

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-219.63}$.

A.6.3.2 ees761ep1

This parameter set is suitable for use at the 128-bit security level

Table 11 – ees761ep1
N = 761
p = 3
q = 2048
Key generation: KGP-3 with
df = 42
dg = 253
lLen = 1
db = 128
maxMsgLenBytes = 125
bufferLenBits = 1136
bufferLenTrits = 760
dm0 = 42
MGF-TP-1 with
SHA-1 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-1 (IGF)
dr = 42
c = 12
minCallsR = 13
minCallsMask = 16
OID = 00 03 05
pkLen = 128

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-258.64}$.

A.6.3.3 ees1087ep1

This parameter set is suitable for use at the 192-bit security level

Table 12 – ees1087ep1
N = 1087
p = 3
q = 2048
Key generation: KGP-3 with
df = 63
dg = 362
lLen = 1
db = 192
maxMsgLenBytes = 178
bufferLenBits = 1624
bufferLenTrits = 1086
dm0 = 63
MGF-TP-1 with
SHA-256 (MGF)
BPGM3 with
IGF-MGF-1 with SHA-256 (IGF)
dr = 63
c = 13
minCallsR = 13
minCallsMask = 14
OID = 00 05 05
pkLen = 192

NOTE— If a message representative m’ has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m’ is $2^{-357.90}$.

A.6.3.4 ees1499ep1

This parameter set is suitable for use at the 256-bit security level

Table 13 – ees1499ep1

Table 13 – ees1499ep1

- N = 1499
- p = 3
- q = 2048
- Key generation: KGP-3 with
  - df = 79
  - dg = 499
- lLen = 1
- db = 256
- maxMsgLenBytes = 247
- bufferLenBits = 2240
- bufferLenTrits = 1498
- dm0 = 79
- MGF-TP-1 with
  - SHA-256 (MGF)
- BPGM3 with
  - IGF-MGF-1 with SHA-256 (IGF)
  - dr = 79
  - c = 13
  - minCallsR = 17
  - minCallsMask = 19
- OID = 00 06 05
- pkLen = 256

NOTE— If a message representative m' has fewer than dm0 1s, -1s, or 0s, it must be rejected. The chance of this happening with a legimitately generated m' is $2^{-440.09}$.

A.7 Security levels of Parameter Sets

A.7.1 Assumed security levels versus current knowledge

These security considerations have noted several places where the assumptions used to generate the parameter sets are more cautious than the best attacks that are currently known. As a result of this, the parameter sets given in this standard for use with a certain security level *k* would in fact have a security level *k'* >*k* against an attacker using the best techniques known in July 2008. This section summarizes the assumptions that have been made that favour the attacker, and compares the known July 2008 security levels of the parameter sets with the security levels for which those parameter sets are recommended.

Area	**Current experimental strength**	**Assumed strength**
Lattice reduction time	$t = 0.4750/\delta + 3\ln(1/\delta) - 123.58$	$t = 0.2/\delta + 3\ln(1/\delta) - 50$
Combinatorial search time for c1 1s, c2 -1s in a space of size K	$\dfrac{\binom{K}{c_1/2}\binom{K-c_1/2}{c_2/2}}{\sqrt{p_s\binom{c_1}{c_1/2}\binom{c_2}{c_2/2}}}$	$\dfrac{\binom{K}{c_1/2}\binom{K-c_1/2}{c_2/2}}{p_s\binom{c_1}{c_1/2}\binom{c_2}{c_2/2}}$
Time to perform Babai reduction	N^2	N
P_{split}	$P_{split,1} = \dfrac{\binom{N-K}{d_1-c_1}\binom{N-K-(d_1-c_1)}{d_2-c_2}\cdot\binom{K}{c_1}\binom{K-c_1}{c_2}}{\binom{N}{c_1}\binom{N-c_1}{c_2}}$	$P_{split,N} = 1 - (1 - P_{split,1})^N$.

Table 14 Assumptions used to generate parameters in this standard vs current best known attacks

Parameter set	**N**	**q**	**df**	**Known strength**	**Recommended security level**
ees401ep1	401	2048	113	154.88	112
ees541ep1	541	2048	49	141.766	112
ees659ep1	659	2048	38	137.861	112
ees449ep1	449	2048	134	179.899	128
ees613ep1	613	2048	55	162.385	128
ees761ep1	761	2048	42	157.191	128
ees653ep1	653	2048	194	276.736	192
ees887ep1	887	2048	81	245.126	192
ees1087ep1	1087	2048	63	236.586	192
ees853ep1	853	2048	268	376.32	256
ees1171ep1	1171	2048	106	327.881	256
ees1499ep1	1499	2048	79	312.949	256

Table 15 Strengths of recommended parameter sets in this standard vs best current attacks

A.7.2 Potential research

As detailed above, the parameter sets in this standard are designed to be secure against incremental improvements in attack techniques. As these improvements occur, future versions of the standard will track the "current known" strength of each parameter set as it descends towards the recommended security level.

There are potential breakthroughs in research that have not been considered in generating these parameter sets, because it is not clear that these breakthroughs will ever come. Such breakthroughs, which would require an in-depth re-evaluation of the security of the algorithm, include:

— Improvement in lattice reduction techniques for the hybrid case beyond the current extrapolation line

— A sub-exponential or otherwise massively improved attack on the whole NTRU lattice

— An improvement in the reduction step of the meet-in-the-middle phase of the hybrid attack that would allow an attacker to significantly increase p_s.

Annex B

(informative)

Bibliography

[B1] M. Ajtai, The shortest vector problem in L2 is NP-hard for randomized reductions, in Proc. of 30th STOC, ACM, 1998.

[B2] M. Ajtai, C. Dwork, A public-key cryptosystem with worst case/average case equivalence. In Proc. 29th ACM Symposium on Theory of Computing, 1997, 284-293.

[B3] M. Ajtai, R. Kumar, D. Sivakumar, A sieve algorithm for the shortest lattice vector problem, 33rd ACM Symposium on Theory of Computing, 2001.

[B4] ANSI INCITS 4-1986 (R2002), *Information Systems — Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII).*

[B5] ANS X9.42-2001, *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*.

[B6] ANS X9.52-1998, *Triple Data Encryption Algorithm Modes of Operation*.

[B7] ANS X9.63-2002, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

[B8] ANS X9.71-2000, *Keyed Hash Message Authentication Code (MAC).*

[B9] L. Babai, *On Lovasz lattice reduction and the nearest lattice point problem*, Combinatorica, vol.~6, 1986, 1--13

[B10] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, pp. 26 – 45. Springer Verlag, 1998.

[B11] S. Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols Proceedings of the 5th Annual Workshop on Selected Areas in Cryptography (SAC '98), Lecture Notes in Computer Science, 1556 (1999), 339-361.

[B12] J. Blömer, J.-P. Seifert, On the Complexity of Computing Short Linearly Independent Vectors and Short Bases in a Lattice, STOC '99

[B13] J. Buchmann, C. Ludwig, Cryptology ePrint Archive Report 2005/072: Practical Lattice Basis Sampling Reduction.

[B14] J.-Y. Cai, Some recent progress on the complexity of lattice problems, in Proc. FCRC, 1999.

[B15] J.-Y. Cai, The complexity of some lattice problems, in Algorithmic Number Theory – Proceedings of ANTS IV, Leiden, W. Bosma, ed., Lecture Notes in Computer Science, Springer-Verlag.

[B16] J.-Y. Cai, T.W. Cusick, A lattice-based public key cryptosystem, Information and Computation 151 (1999), 17-31.

[B17] J.-Y. Cai, A.P. Nerukar, An improved worst-case to average-case reduction for lattice problems, Proc. 38th Symposium on Foundations of Computer Science, 1997, 468-477

[B18] Consortium for Efficient Embedded Security, Efficient Embedded Security Standard (EESS) #1 (http://www.ceesstandards.org).

[B19] D. Coppersmith, A. Shamir, "Lattice Attacks on NTRU", *Advances in Cryptology — Eurocrypt '97*, Lecture Notes in Computer Science 1233, Springer-Verlag, 1997, 52-61.

[B20] Dinur, G. Kindler, S. Safra, Approximating CVP to within almost-polynomial factors is NP-hard, Proc. 39th Symposium on Foundations of Computer Science, 1998, 99-109

[B21] P. van Emde Boas, Another NP-complete problem and the complexity of computing short vectors in a lattice, Technical Report, Mathematische Instuut, University of Amsterdam, 1981.

[B22] Roger Fischlin, Jean-Pierre Seifert. Tensor-based trapdoors for CVP and their applications to public key cryptography, in Cryptography and Coding, Lecture Notes in Computer Science 1746, Spring-Verlag, 1999, 244-257.

[B23] C. Gentry, Key Recovery and Message Attacks on NTRU-Composite, *Proc. EUROCRYPT 2001*, Lecture Notes in Computer Science, Springer-Verlag, 2001

[B24] O. Goldreich, S. Goldwasser, On the limits of non-approximability of lattice problems, Proc. 39th Symposium on Foundations of Computer Science, 1998, 1-9

[B25] O. Goldreich, S. Goldwasser, S. Halvei, Public-key cryptography from lattice reduction problems. In Proc. CRYPTO'97, Lect. Notes in Computer Science 1294, Springer-Verlag, 1997, 112-131.

[B26] O. Goldreich, D. Micciancio, S. Safra, J.-P. Seifert, Approximating shortest lattice vectors is not harder than approximating closest lattice vectors, Electronic Colloquium on Computational Complexity, TR99-002, 1999

[B27] M. Gruber, C.G. Lekkerkerker, Geometry of Numbers, North-Holland, 1987.

[B28] J. Håstad. Solving Simultaneous Modular Equations of Low Degree. *SIAM Journal of Computing*, 17, pp. 336 – 341. 1988.

[B29] C. Heckler, L. Thiele, Complexity analysis of a parallel lattice basis reduction algorithm, Siam J. Comput. 27 (1998), 1295-1302.

[B30] P. Hirschhorn, J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, W. Whyte, Hybrid Lattice reduction and Meet in the Middle Resistant Parameter Selection for NTRUEncrypt, preprint.

[B31] J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: A new high speed public key cryptosystem, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423, J.P. Buhler (ed.), Springer-Verlag, Berlin, 1998, 267—288

[B32] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J.H. Silverman, W. Whyte, NTRUSign: Digital Signatures in the NTRU Lattice, CT-RSA 2003.

[B33] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J.H. Silverman, W. Whyte, Hybrid lattice reduction and meet-in-the-middle resistant parameter selection for NTRU. Preprint, available from http://grouper.ieee.org/groups/1363/lattPK/submissions.html#2007-02.

[B34] J. Hoffstein, J.H. Silverman, Optimizations for NTRU, Public-Key Cryptography and Computational Number Theory (Warsaw, September 11-15, 2000), DeGruyter, to appear.

[B35] J. Hoffstein, J.H. Silverman, Random Small Hamming Weight Products with Applications to Cryptography, Com2MaC Workshop on Cryptography (Pohang, Korea, June 2000), Discrete Mathematics, to appear.

[B36] J. Hoffstein, J. Silverman, W. Whyte, NTRU Technical Report #12, v2, Estimating Breaking Times for NTRU Lattices. Available from http://www.ntru.com/cryptolab/tech_notes.htm#012.

[B37] J. Hong, J. W. Han, D. Kwon, D. Han, Chosen-Ciphertext Attacks on Optimized NTRU, available from http://eprint.iacr.org/2002/188/.

[B38] N. Howgrave-Graham, A Hybrid lattice reduction and meet-in-the-middle-attack against NTRU. Crypto 2007.

[B39] N. Howgrave-Graham, Isodual Reduction of Lattices. Preprint, available from http://eprint.iacr.org/2007/105

[B40] N. Howgrave-Graham, J. Hoffstein, J. Pipher, W. Whyte, On estimating the lattice security of NTRU, available from http://www.ntru.com/cryptolab/articles.htm and http://eprint.iacr.org/2005/104.

[B41] N. Howgrave-Graham, P. Nguyen, D. Pointcheval, J. Proos, A. Singer, W. Whyte, The Impact of Decryption Failures on the Security of NTRU Encryption, available from http://www.ntru.com/cryptolab/articles.htm

[B42] N. Howgrave-Graham, J.H. Silverman, A. Singer, W. Whyte, Modified Parameter Attacks: Practical Attacks Against CCA2 Secure Cryptosystems, and Countermeasures. Preprint available from http://eprint.iacr.org.

[B43] N. Howgrave-Graham, J.H. Silverman, W. Whyte, A meet-in-the-middle attack on an NTRU private key, NTRU Technical Report 004, version 2, 2003. Available from http://www.ntru.com/cryptolab/tech_notes.htm#004.

[B44] N. Howgrave-Graham, J.H. Silverman, W. Whyte, Choosing Parameter Sets for NTRUEncrypt with SVES-3 and NAEP, CT-RSA 2005.

[B45] N. Howgrave-Graham, J.H. Silverman, A. Singer, W. Whyte, Decryption Failures and Provability: SAEP$^+$, NAEP and NTRU, available from http://www.ntru.com/cryptolab/articles.htm

[B46] Richard Hughes, Gary Doolen, David Awschalom, Carlton Caves, Michael Chapman, Robert Clark, David Cory, David DiVincenzo, Artur Ekert, P. Chris Hammel, Paul Kwiat, Seth Lloyd, Gerard Milburn, Terry Orlando, Duncan Steel, Umesh Vazirani, Birgitta Whaley, David Wineland, "A Quantum Information Science and Technology Roadmap, Part 1: Quantum Computation", Report of the Quantum Information Science and Technology Experts Panel, Version 2.0, April 2, 2004, Advanced Research and Development Activity, http://qist.lanl.gov/pdfs/qc_roadmap.pdf

[B47] IEEE Std 1363-2000, *Standard Specifications for Public Key Cryptography*. IEEE, 2000.

[B48] IEEE Std 1363a-2004, *Standard Specifications for Public Key Cryptography: Additional Techniques*. IEEE, 2004.

[B49] ISO/IEC 8824-1:2002. *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*. Also published as ITU-T Recommendation X.680 (2002).

[B50] ISO/IEC 8824-2:2002, *Information technology — Abstract Syntax Notation One (ASN.1): Information object specification*. Also published as ITU-T Recommendation X.681 (2002).

[B51] ISO/IEC 8824-3:2002, *Information technology — Abstract Syntax Notation One (ASN.1): Constraint specification*. Also published as ITU-T Recommendation X.682 (2002).

[B52] ISO/IEC 8824-4:2002, *Information technology — Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*. Also published as ITU-T Recommendation X.683 (2002).

[B53] 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)*. Also published as ITU-T Recommendation X.690 (2002).

[B54] ISO/IEC 8825-2:2002, *Information technology — ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*. Also published as ITU-T Recommendation X.691 (2002).

[B55] ISO/IEC 8825-3:2002, *Information technology — ASN.1 encoding rules: Specification of Encoding Control Notation (ECN)*. Also published as ITU-T Recommendation X.692 (2002).

[B56] ISO/IEC 8825-4:2002, *Information technology — ASN.1 encoding rules: XML Encoding Rules (XER)*. Also published as ITU-T Recommendation X.693 (2002).

[B57] É. Jaulmes, A. Joux, “A chosen-ciphertext attack against NTRU”, *Advances in Cryptology-CRYPTO 2000*, Lecture Notes in Computer Science, Springer-Verlag, 2000

[B58] A. Joux, J. Stern, Lattice reduction: A toolbox for the cryptanalyst, J. of Cryptology 11 (1998), 161-185.

[B59] R. Kannan, Improved algorithms for integer programming and related lattice problems, in Proc. of 15th STOC, 1983, ACM, 193-206.

[B60] R. Kannan, Algorithmic geometry of numbers, Annual review of computer science 2 (1987), 231-267.

[B61] R. Kannan, Minkowski's convex body theorem and integer programming, Math. Oper. Res. 12 (1987), 415-440.

[B62] P. Klein, Finding the closest lattice vector when it's unusually close, in Proc. of SODA 2000, ACM-SIAM, 2000.

[B63] H. Koy, C.P. Schnorr, Segment LLL-reduction of lattice bases, Proceedings of Cryptography and Lattices Conference (CaLC 2001), Lecture Notes in Computer Science, Springer-Verlag.

[B64] H. Koy, C.P. Schnorr, Segment LLL-reduction with floating point orthogonalization, Proceedings of Cryptography and Lattices Conference (CaLC 2001), Lecture Notes in Computer Science, Springer-Verlag.

[B65] H. Krawczyk, M. Bellare and R. Canetti. *IETF RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. February 1997.

[B66] Greg Kuperberg, "A sub-exponential-time quantum algorithm for the dihedral hidden subgroup problem?", 2003, http://arxiv.org/abs/quant-ph/0302112

[B67] J. Lagarias, H.W. Lenstra, C.P. Schnorr, Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice, Combitorica 10 (1990), 333-348

[B68] B. LaMacchia, PhD Thesis, MIT, 1996.

[B69] A.K. Lenstra, H.W. Lenstra, L. Lovasz, "Factoring polynomials with polynomial coefficients", Math. Annalen 261 (1982) 515-534

[B70] A.K. Lenstra, E.R. Verheul, Selecting Cryptographic Key Sizes, Journal of Crytology vol. 14, no. 4, 2001, 255-293.

[B71] C. Ludwig: A Faster Lattice Reduction Method Using Quantum Search, TU-Darmstadt Cryptography and Computeralgebra Technical Report No. TI-3/03, revised version published in Proc. of ISAAC 2003

[B72] A. May, *Auf Polynomgleichungen basierende Public-Key-Kryptosysteme*, Johann Wolfgange Goethe-Universitat, Frankfurt am Main, Fachbereich Informatik. (Masters Thesis in Computer Science, 4 June, 1999; Thesis advisor, Dr. C.P. Schnorr) Available at: www.mi.informatik.uni-frankfurt.de/research/mastertheses.html

[B73] A. May, J.H. Silverman, 'Dimension reduction methods for convolution modular lattices", Cryptography and Lattices Conference (CaLC 2001), Lecture Notes in Computer Science 2146, Springer-Verlag, 2001

[B74] T. Meskanen, A. Renvall, A Wrap Error Attack Against NTRUEncrypt, University of Turku Technical Report TUCS 507, available from http://www.tucs.fi/Research/Series/techreports/techrep.php?year=2003

[B75] D. Miciancio, The shortest vector in a lattice is NP-hard to approximate to within some constant, Proc. 39th Symposium on Foundations of Computer Science, 1998, 92-98.

[B76] M. Naslund, I. Shparlinski, W. Whyte, On the Bit Security of NTRUEncrypt, *Proc. Intern. Workshop on Public Key Cryptography, PKC'03*, Miami, USA, 2003, Lect. Notes in Comp. Sci., Springer-Verlag, Berlin, 2003, v.2567, 62-70. Available from http://www.ntru.com/cryptolab/articles.htm#004.

[B77] National Institute of Standards and Technology (NIST). *AES Key Wrap Specification*. Draft, December 3, 2001. Available at http://csrc.nist.gov/encryption/kms/key-wrap.pdf.

[B78] National Institute of Standards and Technology (NIST). *Special Publication 800-57, Recommendation for Key Management, Part 1: General Guideline*. Draft, January 2003. Available from http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html.

[B79] National Institute of Standards and Technology (NIST). *Special Publication 800-56: Recommendation on Key Establishment Schemes.* Draft 2.0, January 2003. Available from http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html.

[B80] P. Nguyen, Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97, Advances in Cryptology - Proceedings of CRYPTO '99, (August 15-19, 1999, Santa Barbara, California), M. Wiener (ed.), Lecture Notes in Computer Science, Springer-Verlag.

[B81] P. Nguyen, D. Pointcheval, Analysis and Improvements of NTRU Encryption Paddings, Proc. *CRYPTO 2002*, Lecture Notes in Computer Science, Springer-Verlag 2002.

[B82] P. Nguyen, D. Stehle, Floating-point LLL Revisited, Proc. of EUROCRYPT '05.

[B83] P. Nguyen, J. Stern, Lattice Reduction in Cryptology: An Update, *Conference on Lattices and Cryptography (CaLC 2001)*, Lecture Notes in Computer Science 2146, Springer-Verlag

[B84] P. Nguyen, J. Stern, The orthogonal lattice: A new tool for the cryptanalyst, preprint 2001.

[B85] J. Proos, Imperfect Decryption and an Attack on the NTRU Encryption Scheme, available from http://eprint.iacr.org/2003/002/.

[B86] O. Regev, "Quantum computation and lattice problems?", Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science, (IEEE Computer Society Press, Los Alamitos, California, USA, 2002), pp. 520?-530. http://citeseer.ist.psu.edu/regev03quantum.html

[B87] O. Regev, "A Sub-Exponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space?", June 2004, http://arxiv.org/abs/quant-ph/0406151

[B88] J. Roch, G. Villard, Parallel gcd and lattice basis reduction, in Proc. CONPAR92, Lyon, Lecture Notes in Computer Science 634, Springer-Verlag, 1992, 557-564.

[B89] C.-P. Schnorr, A hierarchy of polynomial time lattice basis reduction algorithms, Theoretical Computer Science 53 (1987), 201-224.

[B90] C.P. Schnorr , M. Euchner, Proc. Fundamentals of computation theory, LNCS 529, pages 68-85, 1991

[B91] C.P. Schnorr, H.H. Hoerner, "Attacking the Chor-Rivest crypto-system by improved lattice reduction", Proc. Eurocrypt 1995, LNCS 921, 1-12, 1995

[B92] C.P. Schnorr, "Lattice Reduction by Random Sampling and Birthday Methods", Proceedings STACS 2003, Eds. H. Alt, M. Habib, Springer-Verlag, LNCS 2607, pages 145-156

[B93] V. Shoup. OAEP Reconsidered. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001*, pp. 239 – 259. Springer Verlag, 2001.

[B94] V. Shoup. *A Proposal for an ISO Standard for Public Key Encryption (Version 2.1)*. Manuscript, December 20, 2001. Available from http://shoup.net/papers/.

[B95] V. Shoup, *NTL: a Number Theory Library*, available from http://www.shoup.net.

[B96] J.H. Silverman, Invertibility in truncated polynomial rings, NTRU Technical Report 009, 1998, http://www.ntru.com.

[B97] J. H. Silverman, W. Whyte, Estimating Decryption Failure Probabilities for NTRUEncrypt, available from http://www.ntru.com/cryptolab/articles.htm

[B98] J. H. Silverman, W. Whyte, Timing Attacks on NTRUEncrypt via variation in the number of hash calls, NTRU Technical Report 021, 2007, available from http://www.ntru.com/cryptolab/articles.htm.

[B99] R.D. Silverman. *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths*. *RSA Laboratories' Bulletin* No. 13, April 2000. Available from http://www.rsasecurity.com.rsalabs/bulletins/.

[B100] Tsukiji Tatsuie, Kamiyama Hiroaki, "Efficient algorithm for the unique shortest lattice vector problem using quantum oracle?", IEIC Technical Report (Institute of Electronics, Information and Communication Engineers), VOL.101;NO.44(COMP2001 5-12);PAGE.9-16(2001).

[B101] G. Villard, Parallel lattice basis reduction, in Proc. International Symposium on Symbolic and Algebraic Computation, Berkeley, ACM Press, 1992, 269-277.

[B102] D. Wagner, A Generalized Birthday Problem. In Proceedings of Crypto 2002. Avaialble from http://www.cs.berkeley.edu/~daw/papers/genbday.html