INTERNATIONAL STANDARD

ISO/IEC 29150

First edition
2011-12-15

Information technology — Security techniques — Signcryption

Technologies de l'information — Techniques de sécurité — Signcryptage

Reference number
ISO/IEC 29150:2011(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword v

Introduction vi

1 Scope 1

2 Normative references 1

3 Terms and definitions 2

4 Symbols and notations 7

5 Finite fields and elliptic curves 8
5.1 Finite fields 8
5.2 Elliptic curves 9

6 Conversion functions 10
6.1 Bits and strings 10
6.2 Conversion between bit strings and integers 11
6.3 Conversion between finite field elements and integers/bit strings 11
6.4 Conversion between points on elliptic curves and bit strings 11

7 Cryptographic transformations 12
7.1 Introduction 12
7.2 Cryptographic hash functions 12
7.2.1 Standard cryptographic hash functions 12
7.2.2 Full domain cryptographic hash functions 12
7.2.2.1 General 12
7.2.2.2 Allowable full domain cryptographic hash function (FDH1) 13
7.3 Key derivation functions 13

8 General model for signcryption 13

9 Discrete logarithm based signcryption mechanism (DLSC) 15
9.1 Introduction 15
9.2 Specific requirements 15
9.3 System wide parameters 15
9.4 Key generation algorithm 16
9.5 Signcryption algorithm 16
9.6 Unsigncryption algorithm 17

10 Elliptic curve based signcryption mechanism (ECDLSC) 18
10.1 Introduction 18
10.2 Specific requirements 18
10.3 System wide parameters 18
10.4 Key generation algorithm 19
10.5 Signcryption algorithm 19
10.6 Unsigncryption algorithm 20

11 Integer factorization based signcryption mechanism (IFSC) 21
11.1 Introduction 21
11.2 Specific requirements 22
11.3 System wide parameters 22
11.4 Key generation algorithm 22
11.5 Signcryption algorithm 22
11.6 Unsigncryption algorithm 23

12 Encrypt-then-sign-based mechanism (EtS) 26
12.1 Introduction 26

12.2 Specific requirements **26**
12.3 Key generation algorithm **26**
12.4 Signcryption algorithm **27**
12.5 Unsigncryption algorithm **27**

Annex A (normative) **Object identifiers** **28**

Annex B (informative) **Security considerations** **30**

Annex C (informative) **Guidance on use of the mechanisms** **36**

Annex D (informative) **Examples** **40**

Bibliography **52**

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29150 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

Introduction

When data is sent from one place to another, it is often necessary to protect it in some way whilst it is in transit, e.g. against eavesdropping or unauthorized modification. Similarly, when data is stored in an environment to which unauthorized parties can have access, it is important to protect it against unauthorized access.

If the confidentiality of the data needs to be protected, e.g. against eavesdropping, then one solution is to use public key encryption, as specified in ISO/IEC 18033. Alternatively, if it is necessary to protect the data against unauthorized modification or forgery, then digital signatures, as specified in ISO/IEC 9796 and ISO/IEC 14888, can be used. If both confidentiality and unforgeability are required, then one possibility is to use both public key encryption and digital signature. Whilst these operations can be combined in many ways, not all combinations of such mechanisms provide the same security guarantees. As a result it is desirable to define in detail exactly how confidentiality and unforgeability mechanisms should be combined to provide the optimum level of security. Moreover, in some cases significant efficiency gains can be obtained by defining a single method of processing the data with the objective of providing both confidentiality and unforgeability.

In this International Standard, *signcryption mechanisms* are defined. These are methods for processing data to provide both confidentiality and unforgeability. These data processing methods typically involve either the use of an asymmetric encryption scheme and a digital signature scheme combined in a specific way or the use of a specially developed algorithm which fulfils both functions simultaneously.

The methods specified in this International Standard have been designed to maximise the level of security and provide efficient processing of data. All the mechanisms defined here have mathematical “proofs of security”, i.e. rigorous arguments supporting their security claims.

Information technology — Security techniques — Signcryption

1 Scope

This International Standard specifies four mechanisms for signcryption that employ public key cryptographic techniques requiring both the originator and the recipient of protected data to have their own public and private key pairs.

This International Standard is not applicable to infrastructures for management of public keys which are defined in ISO/IEC 11770-1 and ISO/IEC 9594.

NOTE 1 Signcryption mechanisms are defined ways of processing a data string with the following security objectives:

- **data confidentiality**, i.e. protection against unauthorized disclosure of data;
- **data integrity**, i.e. protection that enables the recipient of data to verify that it has not been modified;
- **data origin authentication**, i.e. protection that enables the recipient of data to verify the identity of the data originator;
- **data unforgeability**, i.e. protection against unauthorized modification of data, even by a recipient of the data.

These four security objectives are not necessarily mutually exclusive. The fourth objective, data unforgeability, in particular is a stronger notion of security that implies both data integrity and data origin authentication.

NOTE 2 Two of the mechanisms specified in this International Standard, namely mechanisms DLSC and ECDLSC, require the employment of system wide public key parameters for both the sender and the recipient of data. In a system where a multiple number of pairs of senders and recipients exist, the same system wide parameters are required to be used by all these users. The two remaining specified mechanisms, namely IFSC and EtS, do not require the use of such system wide public key parameters.

NOTE 3 In selecting the four signcryption mechanisms for inclusion in this International Standard from the large variety of such techniques published and in use, the same seven criteria as those stated in ISO/IEC 18033-1:2005, Annex A, have been followed. The exclusion of particular methods does not imply that those methods are insecure.

NOTE 4 This International Standard bears a conceptual similarity to ISO/IEC 19772[14] which specifies a number of mechanisms for authenticated encryption, that is, simultaneously achieving message integrity and confidentiality. Major differences between ISO/IEC 19772 and this International Standard include (1) mechanisms specified in ISO/IEC 19772 fall into the category of symmetric cryptographic techniques, whereas those specified in this International Standard are representatives of asymmetric cryptographic techniques; (2) while all mechanisms specified in ISO/IEC 19772 and this International Standard offer the capability of data integrity and origin authentication, mechanisms specified in this International Standard further offer the capability of data unforgeability, even by a recipient of the data.

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) applies.

ISO/IEC 9796-2:2010, *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Integer factorization based mechanisms*

ISO/IEC 9796-3:2006, *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 3: Discrete logarithm based mechanisms*

ISO/IEC 14888-1:2008, *Information technology — Security techniques — Digital signatures with appendix — Part 1: General*

ISO/IEC 14888-2:2008, *Information technology — Security techniques — Digital signatures with appendix — Part 2: Integer factorization based mechanisms*

ISO/IEC 14888-3:2006, *Information technology — Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms*

ISO/IEC 18033-1:2005, *Information technology — Security techniques — Encryption algorithms — Part 1: General*

ISO/IEC 18033-2:2006, *Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
asymmetric cipher
alternative term for asymmetric encryption system

[ISO/IEC 18033-1:2005]

3.2
asymmetric cryptographic technique
cryptographic technique that uses two related transformations, a public transformation (defined by the public key) and a private transformation (defined by the private key)

[ISO/IEC 11770-1:2010]

3.3
asymmetric encryption system
system based on asymmetric cryptographic techniques whose public transformation is used for encryption and whose private transformation is used for decryption

[ISO/IEC 9798-1:2010]

3.4
asymmetric key pair
pair of related keys where the private key defines the private transformation and the public key defines the public transformation

[ISO/IEC 9798-1:2010]

3.5
block
string of bits of a defined length

3.6
block cipher
symmetric encryption system with the property that encryption operates on a block of plaintext, i.e. a string of bits of a defined length, to yield a block of ciphertext, and decryption operates on the ciphertext to yield the original plaintext

[ISO/IEC 18033-1:2005]

3.7
cipher
alternative term for encryption system

[ISO/IEC 18033-1:2005]

3.8
ciphertext
data which has been transformed to hide its information content

[ISO/IEC 10116:2006]

3.9
cleartext
alternative term for plaintext

3.10
collision-resistant hash-function
hash-function satisfying the following property: it is computationally infeasible to find any two distinct inputs which map to the same output

[ISO/IEC 10118-1:2000]

3.11
data element
integer or bit string or set of integers or set of bit strings

3.12
decryption
reversal of encryption by a cryptographic algorithm to produce a plaintext

3.13
decryption algorithm
process which transforms a ciphertext into a plaintext

[ISO/IEC 18033-1:2005]

3.14
domain
set of entities operating under a single security policy

[ISO/IEC 14888-1:2008]

3.15
domain parameter
data element which is common to and known by or accessible to all entities within the domain

[ISO/IEC 14888-1:2008]

3.16
encryption
(reversible) transformation of data by a cryptographic algorithm to produce a ciphertext, i.e. to hide the information content of the data

NOTE Adapted from ISO/IEC 9797-1:2011.

3.17
encryption algorithm
process which transforms a plaintext into a ciphertext

[ISO/IEC 18033-1:2005]

3.18
encryption system
cryptographic technique used to protect the confidentiality of data, and which consists of three component processes: a method for generating keys, an encryption algorithm and a decryption algorithm

3.19
full domain cryptographic hash function
function that maps strings of bits to integers in a fixed range, satisfying the properties of (1) for a given output, it is computationally infeasible to find an input which maps to this output, and (2) for a given input, it is computationally infeasible to find a second input which maps to the same output

NOTE A full domain cryptographic hash function is similar to a standard cryptographic hash function with the exception that the former outputs an integer rather than a bit string; see 7.2.2.

3.20
identification data
sequence of data elements, including the distinguishing identifier for an entity, assigned to an entity and used to identify it

NOTE The identification data can additionally contain data elements such as identifier of the signature process, identifier of the signature key, validity period of the signature key, restrictions on key usage, associated security policy parameters, key serial number, or domain parameters.

[ISO/IEC 14888-1:2008]

3.21
key
sequence of symbols that controls the operation of a cryptographic transformation (e.g. encryption, decryption)

[ISO/IEC 11770-1:2010]

3.22
key pair
pair consisting of a public key and a private key associated with an asymmetric cipher

3.23
keystream
pseudorandom sequence of symbols, intended to be secret, used by the encryption and decryption algorithms of a stream cipher

NOTE If a portion of the keystream is known by an attacker, then it is computationally infeasible for the attacker to deduce any information about the remainder of the keystream.

3.24
message
string of bits of any length

3.25
***n*-bit block cipher**
block cipher with the property that plaintext blocks and ciphertext blocks are *n* bits in length

[ISO/IEC 10116:2006]

3.26
one-way hash function
function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:

- for a given output, it is computationally infeasible to find an input which maps to this output;
- for a given input, it is computationally infeasible to find a second input which maps to the same output

[ISO/IEC 10118-1:2000]

3.27
parameter
integer or bit string or function

3.28
plaintext
unencrypted information

[ISO/IEC 10116:2006]

3.29
private key
that key of a key pair associated with an entity's asymmetric cipher which is kept secret and used by that entity only

[ISO/IEC 11770-1:2010]

3.30
public key
that key of a key pair associated with an entity's asymmetric cipher which can be made public and used by any entity

[ISO/IEC 11770-1:2010]

3.31
secret key
key used with symmetric cryptographic techniques by a specified set of entities

[ISO/IEC 11770-3:2008]

3.32
signature
one or more data elements resulting from the signature process

3.33
signature key
set of private data elements specific to an entity and usable only by this entity in the signature process

[ISO/IEC 14888-1:2008]

3.34
signature process
process which takes as inputs the message, the signature key and the domain parameters, and which gives as output the signature

3.35
signcrypt
to apply signcryption on a plaintext

3.36
signcryption
(reversible) transformation of data by a cryptographic algorithm to produce a ciphertext from which no information about the original data can be recovered (except possibly its length), nor can a new ciphertext be forged by an unauthorized entity without detection, that is, it provides data confidentiality, data integrity, data origin authentication, and non-repudiation

NOTE Unforgeability implies data integrity, data origin authentication, and non-repudiation.

3.37
signcryption algorithm
one of the three component algorithms of a signcryption mechanism which takes as input a plaintext, a sender's public and private key pair, a recipient's public key and other data, outputs a ciphertext after performing a sequence of specified operations on the input

3.38
signcryption mechanism
cryptographic technique used to protect the confidentiality and simultaneously guarantee the origin, integrity and non-repudiation of data, and which consists of three component algorithms: a key generation algorithm, a signcryption algorithm and a unsigncryption algorithm

3.39
signed message
set of data elements consisting of the signature, the part of the message which cannot be recovered from the signature, and an optional text field

[ISO/IEC 14888-1:2008]

3.40
symmetric cipher
cipher based on symmetric cryptographic techniques that uses the same secret key for both the encryption and decryption algorithms

[ISO/IEC 18033-1:2005]

3.41
symmetric cryptographic technique
cryptographic technique that uses the same secret key for both the originator's and the recipient's transformation

NOTE 1 Without knowledge of the secret key, it is computationally infeasible to compute either the originator's or the recipient's transformation.

NOTE 2 Examples of symmetric cryptographic techniques include symmetric ciphers and Message Authentication Codes (MACs). In a symmetric cipher, the same secret key is used to encrypt and decrypt data. In a MAC, the same secret key is used to generate and verify MACs.

3.42
unsigncrypt
to apply unsigncryption on a ciphertext

3.43
unsigncryption
verification and decryption of a ciphertext by a cryptographic algorithm

3.44
unsigncryption algorithm
one of the three component algorithms of a signcryption mechanism which takes as input a ciphertext, a recipient's public and private key pair, a sender's public key and other data, outputs a pair consisting of either a symbolic value ACCEPT and a plaintext, or a symbolic value REJECT and the null string

3.45
verification key
set of public data elements which is mathematically related to an entity's signature key and which is used by the verifier in the verification process

[ISO/IEC 14888-1:2008]

3.46
verification process
process which takes as input the signed message, the verification key and the domain parameters, and which gives as output the result of the signature verification: valid or invalid

[ISO/IEC 14888-1:2008]

4 Symbols and notations

For the purposes of this International Standard, the following symbols and notations apply:

$\lfloor x \rfloor$	the largest integer less than or equal to real number x. For example, $\lfloor 8 \rfloor = 8$, $\lfloor 8.7 \rfloor = 8$ and $\lfloor -10.4 \rfloor = -11$.
$\lceil x \rceil$	the smallest integer greater than or equal to real number x. For example, $\lceil 8 \rceil = 8$, $\lceil 8.2 \rceil = 9$, and $\lceil -9.5 \rceil = -9$.
$[a,...,b]$	the interval of integers from a to b, including both a and b.
$(a,...,b)$	the interval of integers from a to b, but excluding both a and b.
$\lvert X \rvert$	if X is a finite set, then the cardinality of X, namely the number of elements in the set X; if X is a finite abelian group or a finite field, then the cardinality of the underlying set of elements; if X is a real number, then the absolute value of X; if X is a bit string, then the length in bits of the string.
$x \oplus y$	the bit-wise exclusive-or (XOR) of two bit strings x and y, where x and y are of equal length. (See also 6.1.)
$\langle x_1,\ldots,x_l \rangle$	the bit string of length l consisting of l bits $x_1,...,x_l$ in the given order. (See also 6.1.)
$x \lVert y$	The result of concatenating two data items x and y in the order specified. In cases where the result of concatenating two or more data items is input to a cryptographic algorithm as part of a signcryption mechanism, this result shall be composed so that it can be uniquely resolved into its constituent data strings, i.e. so that there is no possibility of ambiguity in interpretation. This latter property can be achieved in a variety of different ways, depending on the application. For example, it can be guaranteed by (a) fixing the length of each of the substrings throughout the domain of use of the mechanism, or (b) encoding the sequence of concatenated strings using a method that guarantees unique decoding, e.g. using the distinguished encoding rules defined in ISO/IEC 8825-1 [6].
$\gcd(a,b)$	the greatest common divisor of two integers a and b.
$a \mid b$	integer a divides integer b; that is, there exists an integer c such that $b = ca$.

$a \equiv b \pmod{n}$	integer a and integer b are congruent modulo non-zero integer n; that is $n\|(a-b)$.
$a \mod n$	the unique remainder in $[0,...,n-1]$ when integer a is divided by positive integer n.
$a^{-1} \mod n$	for integer a and positive integer n such that $\gcd(a,n)=1$, the unique integer b in $[1,...,n-1]$ such that $ab \equiv 1 \pmod{n}$.
$L_b(n)$	the length in bits of a non-negative integer n, or the smallest integer l such that $\mathrm{I2BSP}(n,l)$ does not fail; that is, $L_b(n) = \lceil \log_2(n+1) \rceil$, where $\mathrm{I2BSP}(n,l)$, defined in 6.2, converts integer n to a bit string of length l.
AC.Decrypt	decryption algorithm for an asymmetric cipher.
AC.Encrypt	encryption algorithm for an asymmetric cipher.
AC.KeyGen	key generation algorithm for an asymmetric cipher.
ID_X	bit string which uniquely identifies entity X in some context.
pk_d	private decryption key.
pk_s	private signature generation key.
pk_X	private key belonging to the entity X.
$pk_{X,d}$	private decryption key belonging to the entity X.
$pk_{X,s}$	private signature generation key belonging to the entity X.
PK_e	public encryption key.
PK_v	public signature verification key.
PK_X	public key belonging to the entity X.
$PK_{X,e}$	public encryption key belonging to the entity X.
$PK_{X,v}$	public signature verification key belonging to the entity X.
SS.KeyGen	key generation algorithm for a signature scheme.
SS.Sign	signature generation algorithm for a signature scheme.
SS.Verify	signature verification algorithm for a signature scheme.

5 Finite fields and elliptic curves

5.1 Finite fields

This clause describes a very general framework for representing specific finite fields. A finite field specified in this way is called an explicitly given finite field, and it is determined by explicit data.

For a finite field F of cardinality p^e, where p is prime and $e \geq 1$, explicit data for F consists of p and e, along with a "multiplication table" which is a matrix $T = (T_{ij})_{1 \leq i,j \leq e}$, where each T_{ij} is an e-tuple, or an ordered list of e elements, over $[0,..., p-1]$.

The set of elements of F is the set of all e-tuples over $[0,...,p-1]$. The entries of T are themselves viewed as elements of F.

Addition in F is defined element-wise: if $a=(a_1,\ldots,a_e)\in F$ and $b=(b_1,\ldots,b_e)\in F,$ then $a+b=c,$ where

$$c=(c_1,\ldots,c_e) \text{ and } c_i=(a_i+b_i) \bmod p \ (1\le i\le e).$$

A scalar multiplication operation for F is also defined element-wise: if $a=(a_1,\ldots,a_e)\in F$ and $d\in[0,\ldots,p-1]$, then $d\cdot a=c,$ where

$$c=(c_1,\ldots,c_e) \text{ and } c_i=(d\cdot a_i) \bmod p \ (1\le i\le e).$$

Multiplication in F is defined via the multiplication table $T,$ as follows: if $a=(a_1,\ldots,a_e)\in F$ and $b=(b_1,\ldots,b_e)\in F,$ then

$$a\cdot b=\sum_{i=1}^{e}\sum_{j=1}^{e}\left[\left(a_i b_j \bmod p\right)T_{ij}\right]$$

where the products $\left(a_i b_j \bmod p\right)T_{ij}$ are defined using the above rule for scalar multiplication, and where these products are summed using the above rule for addition in $F.$ It is assumed that the multiplication table defines an algebraic structure that satisfies the usual axioms of a field; in particular, there exist additive and multiplicative identities, every element has an additive inverse, and every element besides the additive identity has a multiplicative inverse.

Observe that the additive identity of $F,$ denoted $0_F,$ is the all-zero e-tuple, and that the multiplicative identity of $F,$ denoted 1_F, is a non-zero e-tuple whose precise format depends on $T.$

NOTE 1 The field F is a vector space of dimension e over the prime field F' of cardinality $p,$ where scalar multiplication is defined as above. The prime p is called the characteristic of $F.$ For $1\le i\le e$, let θ_i denote the e-tuple over F' whose i-th component is 1, and all of whose other components are 0. The elements $\theta_1,...,\theta_e$ form an ordered basis of F as a vector space over $F'.$ Note that for $1\le i,j\le e$, we have $\theta_i\cdot\theta_j=T_{ij}$.

NOTE 2 For $e>1,$ two types of standard bases are defined that are commonly used in implementations of finite field arithmetic, namely *polynomial basis* and *normal basis*.

- Polynomial basis: $\theta_1,...,\theta_e$ are called a polynomial basis for F over F' if for some $\theta\in F$, $\theta_i=\theta^{e-i}$ for $1\le i\le e$. Note that in this case, $1_F=\theta_e$.
- Normal basis: $\theta_1,...,\theta_e$ are called a normal basis for F over F' if for some $\theta\in F,$ $\theta_i=\theta^{p^{i-1}}$ for $1\le i\le e$. Note that in this case, $1_F=c\sum_{i=1}^{e}\theta_i$ for some $c\in[1,...,p)$; if $p=2$, then the only possible choice for c is 1; moreover, one can always choose a normal basis for which $c=1$.

NOTE 3 The definition given here of an explicitly given finite field comes from ISO/IEC 18033-2.

5.2 Elliptic curves

An elliptic curve V over an explicitly given finite field F is a set of points $P=(x,y),$ where x and y are elements of F that satisfy a certain equation, together with the "point at infinity" which is denoted by $O.$ For the purposes of this International Standard, the curve V is specified by two field elements $a,b\in F,$ called the coefficients of $V.$

Let p be the characteristic of F. An elliptic curve V over F falls into one of the following three categories, which is determined by the value of p:

- $p = 2$: b shall satisfy $b \neq 0_F$, and every point $P = (x, y)$ on V (other than the point at infinity O) shall satisfy an equation of the form $y^2 + xy = x^3 + ax^2 + b$;
- $p = 3$: a and b shall satisfy $a \neq 0_F$ and $b \neq 0_F$, and every point $P = (x, y)$ on V (other than O) shall satisfy an equation of the form $y^2 = x^3 + ax^2 + b$;
- $p > 3$: a and b shall satisfy $4a^3 + 27b^2 \neq 0_F$, and every point $P = (x, y)$ on V (other than O) shall satisfy an equation of the form $y^2 = x^3 + ax + b$.

Elliptic curves are endowed with the addition operation $+ : V \times V \to V$, defining for each pair (P, Q) of points on V a third point $P + Q$. With respect to this addition, V is an abelian group with identity element O. If $P + Q = O$, then Q is called the additive inverse of P, which is denoted by $-P$. The k-th multiple of a point P is given as kP, where $kP = P + ... + P$ (k summands) if $k > 0$, $kP = (-k)(-P)$ if $k < 0$, and $kP = O$ if $k = 0$. The smallest positive k with $kP = O$ is called the order of P.

There exist efficient algorithms to perform the group operation of an elliptic curve, but the implementation of such algorithms is out of the scope of this International Standard.

A point P (other than O) on an elliptic curve can be represented in compressed, uncompressed, or hybrid form.

If $P = (x, y)$, then (x, y) is the *uncompressed* form of P. The *compressed* form of P is denoted by $(x, \tilde{y})$ where $\tilde{y} \in \{0, 1\}$. The *hybrid* form of $P = (x, y)$ is the triple $(x, \tilde{y}, y)$, where $\tilde{y}$ is as in the compressed form. Given $(x, \tilde{y})$, the compressed form of P, there exist efficient procedures for point *decompression*, i.e., computing y from $(x, \tilde{y})$.

NOTE Information on the implementation of the elliptic curve group operations can be found in ISO/IEC 15946-1.

6 Conversion functions

6.1 Bits and strings

A bit is one of the two symbols ‘0’ and ‘1’. A bit string is a sequence of bits. For l bits $x_1, .., x_l$, the bit string consisting of the l bits, appearing in the given order, is denoted by $\langle x_1, \ldots, x_l \rangle$.

For a bit string $x = \langle x_1, \ldots, x_l \rangle$, the length l of the string is denoted by $|x|$. The left most bit x_1 is called the first, highest order or most significant bit of the string. Likewise, the right most bit x_l is called the last, lowest order or least significant bit of the string.

Given two bit strings $x = \langle x_1, \ldots, x_l \rangle$ and $y = \langle y_1, \ldots, y_m \rangle$, the concatenation of x and y is defined by $x \| y = \langle x_1, ..., x_l, y_1, ..., y_m \rangle$.

For two bit strings of equal length, x and y, their bit-wise exclusive-or (XOR) is denoted by $x \oplus y$.

A null bit string, denoted by NULL, is a string of length 0.

6.2 Conversion between bit strings and integers

$\mathrm{BS2IP}(x)$ takes as input a bit string $x = \langle x_{m-1},...,x_0 \rangle$, and outputs the unique integer value n defined by $n = \sum_{i=0}^{m-1}(v_i 2^i)$, where $v_i = \begin{cases} 0, & \text{if } x_i = \text{'0'} \\ 1, & \text{if } x_i = \text{'1'} \end{cases}$.

$\mathrm{I2BSP}(n,l)$ takes as input two non-negative integers n and l, and outputs the unique bit string x of length l such that $n = \mathrm{BS2IP}(x)$, if such an x exists. Otherwise the function **fails**.

6.3 Conversion between finite field elements and integers/bit strings

$\mathrm{FE2IP}_F(a)$ takes as input an element a of a given finite field F, and outputs an integer value n as follows:

If the cardinality of F is $q = p^m$, where p is prime and $m \geq 1$, then an element a of F is an m-tuple $(a_1,...,a_m)$, where $a_i \in [0,...,p)$ for $1 \leq i \leq m$, and the value n is defined as $n = \sum_{i=1}^{m} a_i p^{i-1}$.

$\mathrm{FE2BSP}_F(a)$ takes as input an element a of a given finite field F, and outputs the bit string $\mathrm{I2BSP}(n,l)$, where $n = \mathrm{FE2IP}_F(a)$ and $l = \lceil \log_2 |F| \rceil$.

$\mathrm{BS2FEP}_F(x)$ takes as input a bit string x, and outputs the unique field element a of F such that $x = \mathrm{FE2BSP}_F(a)$, if such an a exists. Otherwise the function **fails**.

6.4 Conversion between points on elliptic curves and bit strings

$\mathrm{EC2BSP}_V(P, fmt)$ takes as input an element P on a given elliptic curve V, over an explicitly given finite field F, together with a format specifier fmt, which is one of the three symbolic values *compressed*, *uncompressed*, or *hybrid*, and outputs a bit string EP according to rules specified below.

1) If $P = O$, then $EP = \text{'0'}$ (the '0' bit).
2) If $P \neq O$, then computing EP is dictated by the value of fmt. Specifically,
   - 2.1) when $fmt = hybrid$, that is $P = (x, \tilde{y}, y)$, EP is defined as $EP = \mathrm{I2BSP}(6 + \tilde{y}, 3) \,||\, \mathrm{FE2BSP}_F(x) \,||\, \mathrm{FE2BSP}_F(y)$;
   - 2.2) when $fmt = uncompressed$, that is $P = (x, y)$, EP is defined as $EP = \mathrm{I2BSP}(4,3) \,||\, \mathrm{FE2BSP}_F(x) \,||\, \mathrm{FE2BSP}_F(y)$;
   - 2.3) when $fmt = compressed$, that is $P = (x, \tilde{y})$, EP is defined as $EP = \mathrm{I2BSP}(2 + \tilde{y}, 3) \,||\, \mathrm{FE2BSP}_F(x)$.

$\mathrm{BS2ECP}_V(EP)$ takes as input a bit string EP, and outputs a point P, in uncompressed form, on a given elliptic curve V such that $EP = \mathrm{EC2BSP}_V(P, fmt)$ for a format specifier fmt, if such a point P exists. Otherwise the function **fails**.

NOTE 1 L_b is defined in Clause 4.

NOTE 2 Other than FE2BSP, BS2FEP, EC2BSP and BS2ECP, the conversion functions are defined in ISO/IEC 15946-1:2008, Clause 6 (see [12]).

7 Cryptographic transformations

7.1 Introduction

This clause describes several cryptographic transformations that will be referred to in subsequent clauses. The types of transformations are cryptographic hash functions and key derivation functions. For each type of transformation, the abstract input/output characteristics are given, and then specific implementations of these transformations that are allowed for use in this International Standard are specified.

7.2 Cryptographic hash functions

7.2.1 Standard cryptographic hash functions

A (standard) cryptographic hash function is essentially a function that maps a bit string of variable length to a bit string of fixed length. More precisely, a cryptographic hash function *Hash* specifies

- a positive integer *Hash.Len* that denotes the length of the hash function output,
- a positive integer *Hash.MaxInputLen* that denotes the maximum length of the hash function input, and
- a function *Hash.Eval* that denotes the hash function itself, which maps bit strings of length at most *Hash.MaxInputLen* to bit strings of length *Hash.Len*.

The invocation of *Hash.Eval* fails if and only if the input length exceeds *Hash.MaxInputLen*.

When an input to a standard cryptographic hash function consists of several bit strings, these bit strings will be concatenated first, in the order these bit strings are given, to form a single bit string prior to the application of a hash operation.

For the purposes of this International Standard, the allowable standard cryptographic hash functions shall be those described in ISO/IEC 10118-2 and ISO/IEC 10118-3 (see [10]) with the following proviso:

- Whereas the hash functions in ISO/IEC 10118 are not defined for inputs exceeding a given length, a hash function in this International Standard is defined to fail for such inputs.

7.2.2 Full domain cryptographic hash functions

7.2.2.1 General

A full domain cryptographic hash function is a hash function that maps a bit string of arbitrary length to an integer in a fixed range. More precisely, a full domain cryptographic hash function FDH specifies

- a positive integer *FDH.Range* that defines the allowable range of the hash function output, i.e. the hash function outputs an integer in the range of $[0,...,FDH.Range-1]$,
- a positive integer *FDH.MaxInputLen* that denotes the maximum length of the hash function input, and
- a function *FDH.Eval* that denotes the hash function itself, which maps bit strings of length at most *FDH.MaxInputLen* to non-negative integers smaller than *FDH.Range*.

Analogous to a standard cryptographic hash function, when an input to a full domain cryptographic hash function consists of several bit strings, these bit strings will be concatenated first, in the order these bit strings are given, to form a single bit string prior to the application of a hash operation.

The invocation of *FDH.Eval* fails if the length of the hash function input exceeds *FDH.MaxInputLen*. It may also fail in other events that are dependent on specific implementations. An example of such a failure event is when internal operations of *FDH.Eval* transcend a pre-determined maximum allowable number of iterations without producing a valid output.

7.2.2.2 Allowable full domain cryptographic hash function (FDH1)

FDH1 is a family of full domain cryptographic hash functions parameterised by a system parameter denoted by *Hash* which is a (standard) cryptographic hash function, as is described in 7.2.1, with the property that its output length in bits, *Hash.Len*, is at least $\lceil \log_2 FDH1.Range \rceil$.

FDH1 inherits *Hash.MaxInputLen* as its own *FDH1.MaxInputLen* for the maximum length of input bit strings.

Given an input bit string x, $FDH1.Eval(x)$ shall work as follows to produce an integer in the range of $[0,...,FDH1.Range-1]$. It shall indicate failure when the length of x exceeds (*FDH1.MaxInputLen* - 64), or in an extremely rare event when the "while" loop iterates 2^{64} times without bringing forth a valid integer output in the range of $[0,...,FDH1.Range-1]$.

1. If $|x|+64 > FDH1.MaxInputLen$ then FDH1 **fails**.
2. $Counter$ = 0.
3. While $Counter$ < 2^{64} do:
    a. Compute $y = Hash.Eval(x \,||\, \mathrm{I2BSP}(Counter, 64))$;
    b. Set z to the $\lceil \log_2 FDH1.Range \rceil$ left most / higher order bits of y;
    c. If $\mathrm{BS2IP}(z) < FDH1.Range$ then output $\mathrm{BS2IP}(z)$ and quit the procedure;
    d. Increase the value of $Counter$ by 1.
4. FDH1 **fails**.

7.3 Key derivation functions

A key derivation function is a function $KDF(x,l)$ that takes as input a seed x, which is a bit string, and a positive integer l, and outputs a bit string of length l. The string x is of arbitrary length, although an implementation may define a (very large) maximum length for x and maximum size for l, and fail if these bounds are exceeded.

The key derivation functions that are allowed in this International Standard shall be KDF1 and KDF2, as described in ISO/IEC 18033-2, with the following proviso:

- The key derivation functions described in ISO/IEC 18033-2 map octet strings to octet strings, whereas in this International Standard, they map bit strings to bit strings. (An octet is a bit string of length 8.) Mapping between octet strings and bit strings is affected by conversions OS2BSP and BS2OSP which are defined in ISO/IEC 18033-2.

NOTE KDF1 and KDF2 are the same except that the internal counter of KDF1 starts at 0 whereas the internal counter of KDF2 starts at 1.

8 General model for signcryption

A signcryption mechanism SC consists of three algorithms, namely, a key generation algorithm, a signcryption algorithm and an unsigncryption algorithm.

- A key generation algorithm $SC.KeyGen$ outputs a pair of matching public and private keys. The input to the key generation algorithm and the structure of public and private keys are dependent on the particular signcryption mechanism.

- A signcryption algorithm $SC.Signcrypt$ performs a sequence of operations on an input and outputs a ciphertext. The input shall consist of

    1) a plaintext,

    2) a sender's public and private key pair,

    3) a recipient's public key,

    4) a label, and

    5) an option.

    The plaintext, label and ciphertext are all bit strings, whereas the types and structures of public and private keys, as well as that of the option, are determined by the particular signcryption mechanism.

- An unsigncryption algorithm $SC.Unsigncrypt$ performs a sequence of operations on an input and outputs either (ACCEPT, a plaintext) or (REJECT, NULL). The input shall consist of

    1) a ciphertext,

    2) a recipient's public and private key pair,

    3) a sender's public key,

    4) a label, and

    5) an option.

    ACCEPT and REJECT in the output are symbolic values that indicate the acceptance and rejection of validity of the ciphertext in the input, respectively, and NULL is the null string.

Plaintexts may be of variable or fixed length, depending on the particular signcryption mechanism. If the signcryption mechanism can handle plaintexts of variable length, a parameter called $SC.MaxMsgLen$ may be imposed by an implementation to an upper bound on the length of plaintexts. Otherwise if the signcryption mechanism can handle plaintexts of fixed length only, a parameter called $SC.MsgLen$ is employed to define the plaintext length.

Part of the input to a signcryption/unsigncryption algorithm is a label and an option. Their functions are defined as follows.

- A *label* is a (possibly empty) bit string that participates in the signcryption of a plaintext, but need not be protected for confidentiality. An example of a label is a string of public data that is either explicit or implicit from context and required to be bound to the ciphertext. The length of a label may be variable, upper bounded only by an implementation dependant parameter $SC.LabelLen$.

    The same label is required to be used by both the signcryption and unsigncryption algorithms in order for a recipient to correctly unsigncrypt a ciphertext.

    Procedures for agreeing on a label for a specific signcryption mechanism or application are beyond the scope of this International Standard.

- An *option* is a (possibly empty) input argument that passes application and mechanism specific information to signcryption and unsigncryption algorithms. As an example, the elliptic curve based signcryption mechanism defined in this International Standard may use an option to indicate the desired format for encoding points on an elliptic curve. A second possible use of options is to pass system wide parameters specific to a signcryption mechanism. And a third possible use of options is

to pass application or mechanism specific information, such as private factors of composite moduli, to aid faster signcryption or unsigncryption.

An option used by an unsigncryption algorithm may be different from an option used by the signcryption algorithm. While information transmitted by an option may be used in signcryption or unsigncryption, the option itself is not directly involved in the signcryption or unsigncryption process.

Determination of these options for a specific signcryption mechanism or application is beyond the scope of this International Standard.

9 Discrete logarithm based signcryption mechanism (DLSC)

9.1 Introduction

In this clause a discrete logarithm based signcryption mechanism is defined. This mechanism is called DLSC. DLSC can handle plaintexts of variable length, upper bounded only by an implementation dependant parameter $DLSC.MaxMsgLen$.

NOTE DLSC is due to Zheng [23], with small tweaks due to Baek, Steinfeld and Zheng [3]. DLSC possesses security proofs for confidentiality and unforgeability. The security proof for confidentiality relies on the assumption for the Gap Diffie-Hellman problem and the random oracle model, whereas the security proof for unforgeability relies on the assumption for the Gap Discrete Logarithm problem and the random oracle model.

9.2 Specific requirements

Denote by (y_A, x_A) the public and private key pair of the sender, and by (y_B, x_B) the public and private key pair of the recipient. Both key pairs shall be generated*, independently of each other*, using the key generation algorithm as is described in 9.4. Furthermore, any entity that may have to send signcrypted messages and receive signcrypted messages shall have two independently generated key pairs. The first key pair shall be known as the sender key pair that shall only be used to signcrypt a message from that entity. The second key pair shall be known as the recipient key pair that shall only be used to unsigncrypt ciphertexts received by that entity.

9.3 System wide parameters

l_q a positive integer that specifies the length in bits of prime q.

l_p a positive integer that specifies the length in bits of prime p.

q a random prime of l_q bits in length, that is $L_b(q) = l_q$.

p a random prime of l_p bits in length, that is $L_b(p) = l_p$, such that q is a prime factor of $p-1$, that is, $q \mid (p-1)$.

g a random integer from $[2,..., p-1]$ with order q modulo p; that is q is the smallest positive exponent for which $g^q \equiv 1 \bmod p$.

G a key derivation function that takes as input a bit string and an integer, and outputs a bit string of length specified by the integer. (See 7.3)

H a full domain cryptographic hash function that maps an input bit string of arbitrary length to an integer in $[0,...,q-1]$. (See 7.2.2.)

The same algorithms for generating DSA domain parameters, as are described in Annex C of ISO/IEC 14888-3, shall be used to generate p, q and g.

9.4 Key generation algorithm

The key generation algorithm *DLSC.KeyGen* takes as input system wide parameters p, q and g, and runs as follows to output a pair of matching public and private keys.

1. Generate a random integer x from $[1,...,q-1]$.
2. Compute $y = g^x \bmod p$, an integer in $[2,..., p-1]$.
3. Output (y, x) as a public and private key pair.

9.5 Signcryption algorithm

The signcryption algorithm *DLSC.Signcrypt* takes as input a plaintext M (a bit string), a sender's public and private key pair (y_A, x_A), a recipient's public key y_B, a *label* (a bit string), and an *option*. The sender shall perform the following steps to signcrypt the plaintext M. The resultant ciphertext is denoted by X.

1. If y_B is not an integer in the range of $[2,..., p-1]$, then **fail**.
2. Choose a random integer u from $[1,...,q-1]$.
3. Compute $K = y_B^u \bmod p$.
4. Compute $C = G(\text{I2BSP}(K, l_p), |M|) \oplus M$.
5. Compute $r = H(\text{I2BSP}(K, l_p) \,||\, M \,||\, \text{I2BSP}(y_A, l_p) \,||\, \text{I2BSP}(y_B, l_p) \,||\, label)$.
6. If $r + x_A \equiv 0 \pmod q$, then return to Step 2.
7. Compute $s = \dfrac{u}{r + x_A} \bmod q$.
8. Compute $r^* = \text{I2BSP}(r, l_q)$.
9. Compute $s^* = \text{I2BSP}(s, l_q)$.
10. Set $X = (C \,||\, r^* \,||\, s^*)$.
11. Output X.

DLSC.Signcrypt is also illustrated in Figure 1.

NOTE It is assumed that system wide parameters defined in 9.3 are implicitly known to the signcryption algorithm, or otherwise passed to the algorithm by such a means as the *option*.

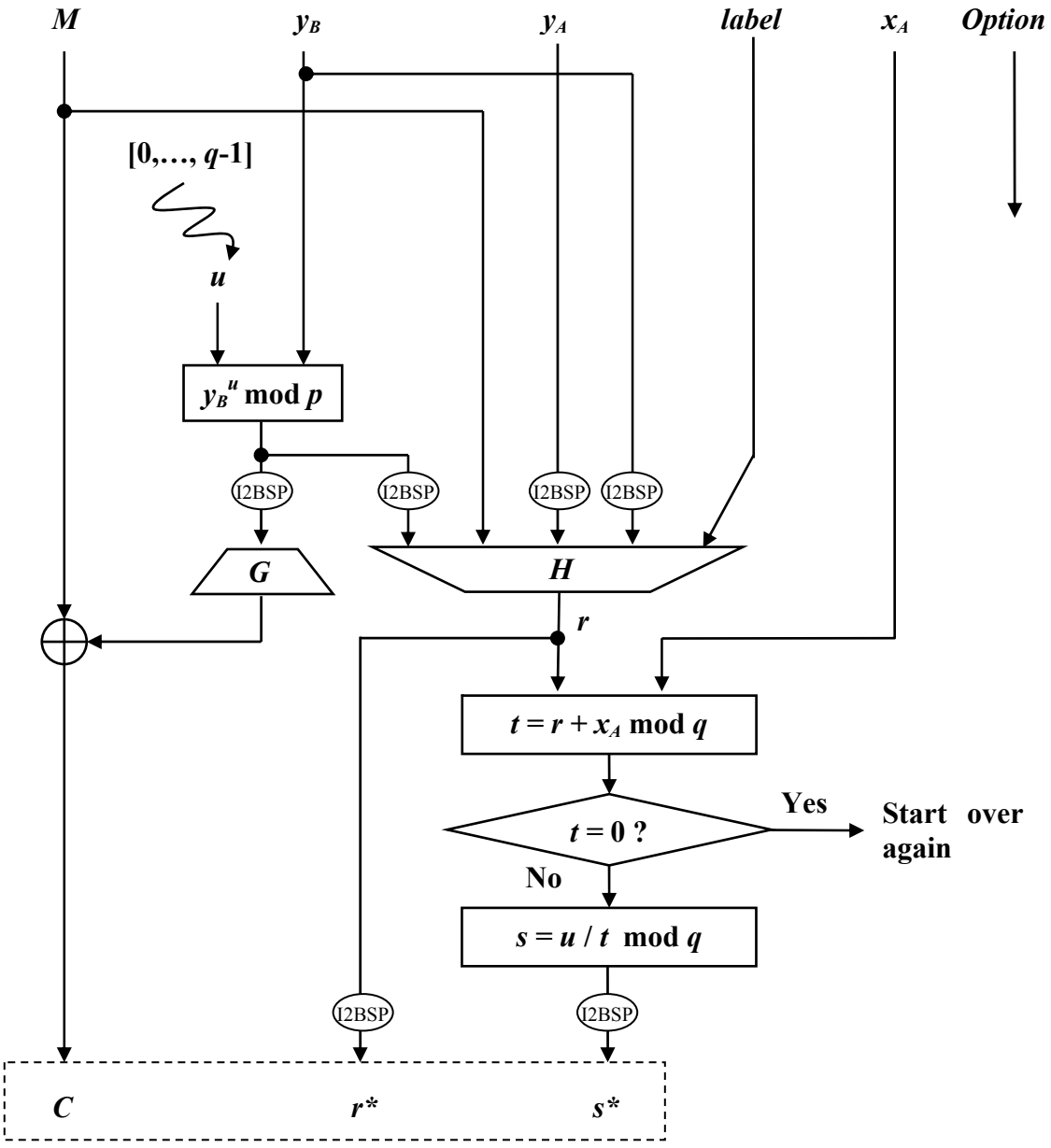

Figure 1 — Signcrypting a plaintext with *DLSC.Signcrypt*

9.6 Unsigncryption algorithm

The unsigncryption algorithm *DLSC.Unsigncrypt* takes as input a ciphertext X (a bit string), a recipient's public and private key pair (y_B, x_B), a sender's public key y_A, a *label* (a bit string), and an *option*. The recipient shall perform the following steps to unsigncrypt the ciphertext X.

1. If y_A is not an integer in the range of $[2,..., p-1]$, then **fail**.
2. Parse X as $(C \| r^* \| s^*)$, where $|r^*| = |s^*| = l_q$.
   If the parsing is unsuccessful, then **fail**.
3. Compute $r = \mathrm{BS2IP}(r^*)$.
4. Compute $s = \mathrm{BS2IP}(s^*)$.

5. If r is not in $[0,...q-1]$ or s is not in $[1,...q-1]$, then **fail**.

6. Compute $K = (g^r \cdot y_A)^{s \cdot x_B} \bmod p$.

7. Compute $M = G(\mathrm{I2BSP}(K, l_p), |C|) \oplus C$.

8. Compute $v = H(\mathrm{I2BSP}(K, l_p) \| M \| \mathrm{I2BSP}(y_A, l_p) \| \mathrm{I2BSP}(y_B, l_p) \| label)$.

9. If $v = r$, then output (ACCEPT, M); otherwise output (REJECT, NULL), where ACCEPT and REJECT are symbolic values that indicate the acceptance and rejection of validity of the ciphertext, respectively, and NULL is the null string.

NOTE It is assumed that system wide parameters defined in 9.3 are implicitly known to the unsigncryption algorithm, or otherwise passed to the algorithm by such a means as the $option$.

10 Elliptic curve based signcryption mechanism (ECDLSC)

10.1 Introduction

In this clause an elliptic curve discrete logarithm based signcryption mechanism is defined. This mechanism is called ECDLSC. ECDLSC can handle plaintexts of variable length, upper bounded only by an implementation dependant parameter $ECDLSC.MaxMsgLen$.

NOTE ECDLSC is due to Zheng and Imai [23][24] with small tweaks due to Baek, Steinfeld and Zheng [3]. ECDLSC possesses security proofs for confidentiality and unforgeability. The security proof for confidentiality relies on the assumption for the Elliptic Curve Gap Diffie-Hellman problem and the random oracle model, whereas the security proof for unforgeability relies on the assumption for the Elliptic Curve Gap Discrete Logarithm problem and the random oracle model.

10.2 Specific requirements

Denote by (Y_A, x_A) the public and private key pair of the sender, and by (Y_B, x_B) the public and private key pair of the recipient. Both key pairs shall be generated, *independently of each other*, using the key generation algorithm as is described in 10.4. Furthermore, any entity that may have to send signcrypted messages and receive signcrypted messages shall have two independently generated key pairs. The first key pair shall be known as the sender key pair that shall only be used to signcrypt a message from that entity. The second key pair shall be known as the recipient key pair that shall only be used to unsigncrypt ciphertexts received by that entity.

10.3 System wide parameters

m	a positive integer (required only if a binary field $GF(2^m)$ is employed).
l_p	a positive integer that specifies the length in bits of prime p (required only if a prime field $GF(p)$ is employed).
p	a prime of l_p bits in length, that is $L_b(p) = l_p$ (required only if a prime field $GF(p)$ is employed).
$GF(p^m)$	a finite field in the form of either $GF(2^m)$ with $m > 1$ (a binary field) or $GF(p)$ with $p > 2$ (a prime field).
V	an elliptic curve over $GF(p^m)$.
l_q	a positive integer that specifies the length in bits of prime q .

q	a prime of l_q bits in length, that is $L_b(q) = l_q$.
J	a designated point on the elliptic curve V with order q. J is also called a base point.
fmt	a format specifier whose value is chosen from {*uncompressed*, *compressed*, *hybrid*}.
G	a key derivation function that takes as input a bit string and an integer, and outputs a bit string of length specified by the integer. (See 7.3)
H	a full domain cryptographic hash function that maps an input bit string of arbitrary length to an integer in $[0,...,q-1]$. (See 7.2.2.)

NOTE Algorithms for generating elliptic curves, finding a point of large prime order on an elliptic curve and others are specified in ISO/IEC 15946-5.

10.4 Key generation algorithm

The key generation algorithm *ECDLSC.KeyGen* takes as input system wide parameters V, q, and J, and runs as follows to output a pair of matching public and private keys.

1. Generate a random integer x from $[1,...,q-1]$.
2. Compute $Y = x \cdot J,$ a point on the elliptic curve V.
3. Output (Y, x) as a public and private key pair.

10.5 Signcryption algorithm

The signcryption algorithm *ECDLSC.Signcrypt* takes as input a plaintext M (a bit string), a sender's public and private key pair (Y_A, x_A), a recipient's public key Y_B, a *label* (a bit string), and an *option*. The sender shall perform the following steps to signcrypt the plaintext M. The resultant ciphertext is denoted by X.

1. If Y_B is not a point on the elliptic curve $V,$ then **fail**.
2. Choose a random integer u from $[1,...,q-1]$.
3. Compute $K = u \cdot Y_B$.
4. Compute $C = G(\mathrm{EC2BSP}_V(K, fmt), |M|) \oplus M$.
5. Compute $r = H(\mathrm{EC2BSP}_V(K, fmt) \,||\, M \,||\, \mathrm{EC2BSP}_V(Y_A, fmt) \,||\, \mathrm{EC2BSP}_V(Y_B, fmt) \,||\, label)$.
6. If $r + x_A \equiv 0 \pmod{q},$ then return to Step 2.
7. Compute $s = \dfrac{u}{r + x_A} \bmod q$.
8. Compute $r^* = \mathrm{I2BSP}(r, l_q)$.
9. Compute $s^* = \mathrm{I2BSP}(s, l_q)$.
10. Set $X = (C \,||\, r^* \,||\, s^*)$.
11. Output X.

ECDLSC.Signcrypt is also illustrated in Figure 2.

NOTE It is assumed that system wide parameters defined in 10.3 are implicitly known to the signcryption algorithm, or otherwise passed to the algorithm by such a means as the $option$. In addition, the format for encoding points on the elliptic curve, fmt, can be passed to the algorithm by the $option$.

10.6 Unsigncryption algorithm

The unsigncryption algorithm *ECDLSC.Unsigncrypt* takes as input a ciphertext X (a bit string), a recipient's public and private key pair (Y_B, x_B), a sender's public key Y_A, a $label$ (a bit string), and an $option$. The recipient shall perform the following steps to unsigncrypt the ciphertext X.

1. If Y_A is not a point on the elliptic curve V, then **fail**.
2. Parse X as $(C \| r^* \| s^*)$, where $|r^*| = |s^*| = l_q$.
   If the parsing is unsuccessful, then **fail**.
3. Compute $r = \mathrm{BS2IP}(r^*)$.
4. Compute $s = \mathrm{BS2IP}(s^*)$.
5. If r is not in $[0,...q-1]$ or s is not in $[1,...q-1]$, then **fail**.
6. Compute $K = s \cdot x_B \cdot (r \cdot J + Y_A)$.
7. Compute $M = G(\mathrm{EC2BSP}_V(K, fmt), |C|) \oplus C$.
8. Compute $v = H(\mathrm{EC2BSP}_V(K, fmt) \| M \| \mathrm{EC2BSP}_V(Y_A, fmt) \| \mathrm{EC2BSP}_V(Y_B, fmt) \| label)$.
9. If $v = r$, then output (ACCEPT, M); otherwise output (REJECT, NULL), where ACCEPT and REJECT are symbolic values that indicate the acceptance and rejection of validity of the ciphertext, respectively, and NULL is the null string.

NOTE It is assumed that system wide parameters defined in 10.3 are implicitly known to the unsigncryption algorithm, or otherwise passed to the algorithm by such a means as the $option$. In addition, the format for encoding points on the elliptic curve, fmt, can be passed to the algorithm by the $option$.

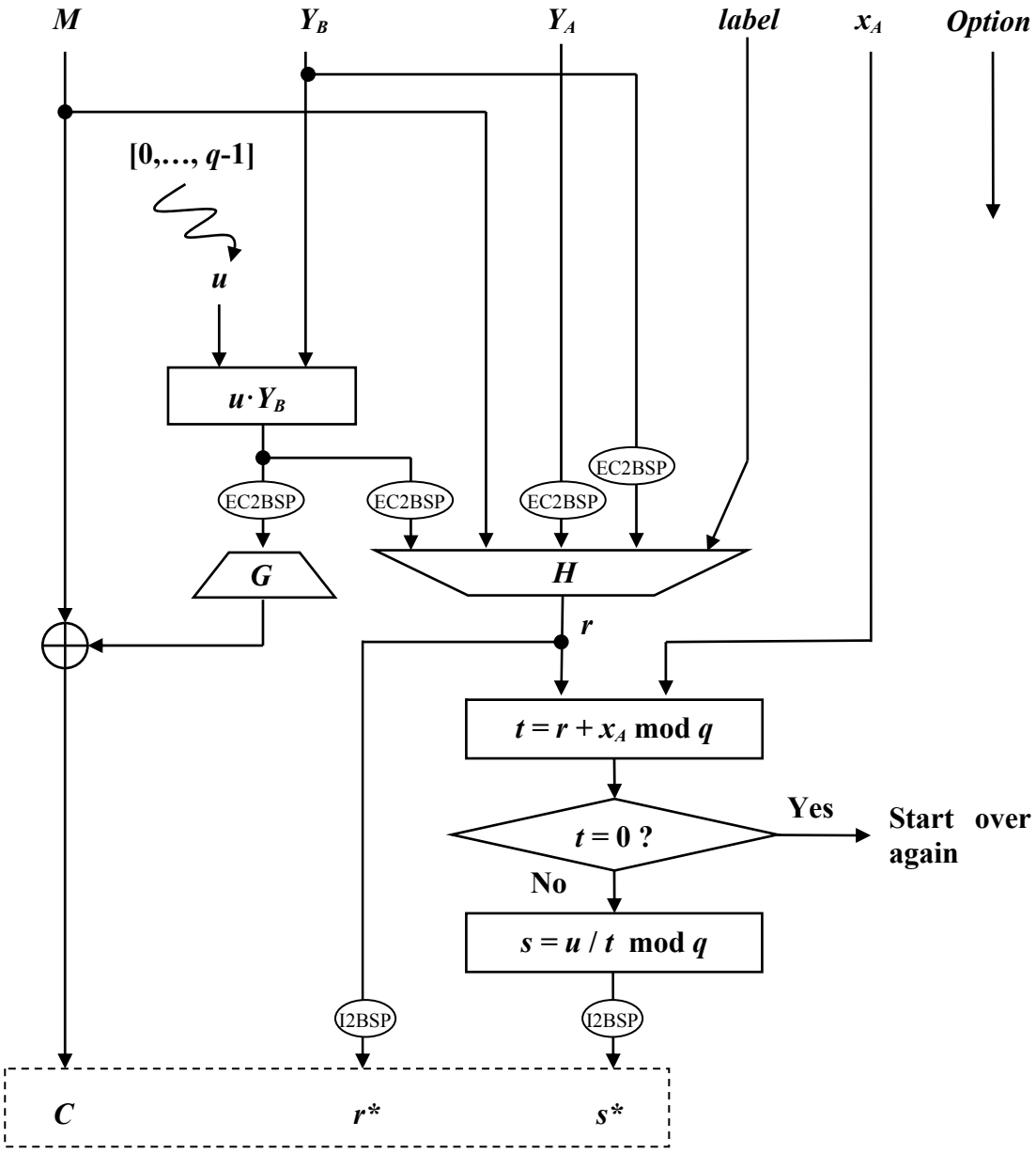

Figure 2 — Signcrypting a plaintext with *ECDLSC.Signcrypt*

11 Integer factorization based signcryption mechanism (IFSC)

11.1 Introduction

In this clause an integer factorization based signcryption mechanism is defined. This mechanism is called IFSC. IFSC can handle plaintexts of fixed length determined by a parameter $IFSC.MsgLen = l_M$.

NOTE IFSC is due primarily to Malone-Lee and Mao [17], with improvements in security by Dodis, Freedman, Jarecki and Walfish [4]. IFSC possesses security proofs for confidentiality and unforgeability. The security proofs rely on the random oracle model and on the assumption that inverting the RSA function is difficult.

11.2 Specific requirements

Denote by $[(N_A, e_A),\ (N_A, d_A)]$ the public and private key pair of the sender, and by $[(N_B, e_B), (N_B, d_B)]$ the public and private key pair of the recipient. Both key pairs shall be generated, *independently of each other*, using the key generation algorithm as is described in 11.4. This mechanism allows an entity to use a single key pair for both signcryption and unsigncryption (i.e. an entity may use the same key pair as both its sender key pair to signcrypt messages originating from that entity and as its recipient key pair to unsigncrypt ciphertexts received by that entity).

11.3 System wide parameters

l_r a positive integer that specifies the length of a random bit string r chosen in the first step of the signcryption algorithm.

l_H a positive integer that specifies the output length in bits of a standard cryptographic hash function H.

l_M a positive integer that specifies the length in bits of a plaintext, that is $IFSC.MsgLen = l_M$.

l a positive even integer that specifies the length in bits of an RSA modulus. l satisfies $l = l_M + l_r + l_H$.

G a key derivation function that takes as input a bit string and an integer, and outputs a bit string of length specified by the integer.

H_1, H_2 two independent (standard) cryptographic hash functions each of which maps an input bit string of arbitrary length to a bit string of length $H_1.Len = H_2.Len = l_H$.

11.4 Key generation algorithm

The key generation algorithm *IFSC.KeyGen* takes a system wide parameter l as input, and outputs a pair of matching public and private keys denoted by $[(N, e), (N, d)]$. These three positive integers N, e and d are constrained by conditions specified below.

- N is the product of two distinct primes p and q of $l/2$ bits in length, that is $L_b(p) = L_b(q) = l/2$. The length of N is l bits.
- e satisfies $\gcd(e, (p-1)(q-1)) = 1$.
- d satisfies $e \cdot d \equiv 1 (\mathrm{mod}\, \lambda(N))$, where $\lambda(N)$ is the least common multiple of $(p-1)$ and $(q-1)$.

The exponent e in the public key is called a *public exponent*, and the exponent d in the private key a *private exponent*.

NOTE 1 Guidance for generating primes p and q can be found in ISO/IEC 18032.

NOTE 2 A relatively small prime such as 5, 17 or $2^{16} + 1 = 65537$ can be selected as the public exponent e for faster signcryption and unsigncryption.

11.5 Signcryption algorithm

The signcryption algorithm *IFSC.Signcrypt* takes as input a plaintext M of l_M bits in length, a sender's private key (N_A, d_A), a recipient's public key (N_B, e_B), a *label* (a bit string), and an *option*. The sender shall perform the following steps to signcrypt the plaintext M. The resultant ciphertext is denoted by X whose length is $l+1$ in bits.

1. Choose uniformly at random a bit string r of length l_r.
2. Compute $c = H_1(M \| r \| label)$.
3. Compute $w = G(c, l_M + l_r) \oplus (M \| r)$.
4. Compute $s = H_2(w) \oplus c$.
5. If $\mathrm{BS2IP}(w \| s) \geq N_A$, return to Step 1.
6. Compute $t = [\mathrm{BS2IP}(w \| s)]^{d_A} \bmod N_A$.
7. If $t \geq N_B$, then set $f = 1$; otherwise set $f = 0$.
8. Compute $u = t - f \cdot 2^{l-1}$.
9. Compute $v = u^{e_B} \bmod N_B$.
10. Compute $C_1 = \mathrm{I2BSP}(f, 1)$.
11. Compute $C_2 = \mathrm{I2BSP}(v, l)$.
12. Set $X = (C_1 \| C_2)$.
13. Output X.

IFSC.Signcrypt is also illustrated in Figure 3.

NOTE It is assumed that system wide parameters defined in 11.3 are implicitly known to the signcryption algorithm, or otherwise passed to the algorithm by such a means as the *option*. The *option* can also be used to pass the prime factors of N_A which are required to speed up the exponentiation with the private exponent d_A in Step 6, using the Chinese Remainder Theorem (see [15], 4.3.2 Modular Arithmetic, [19] and ISO/IEC 14888-2:2008, 5.3).

11.6 Unsigncryption algorithm

The unsigncryption algorithm *IFSC.Unsigncrypt* takes as input a ciphertext X (a bit string), a recipient's private key (N_B, d_B), a sender's public key (N_A, e_A), a *label* (a bit string), and an *option*. The recipient shall perform the following steps to unsigncrypt the ciphertext X.

1. Parse X as $(C_1 \| C_2)$, where $|C_1| = 1$ and $|C_2| = l$.
   If the parsing is unsuccessful, then **fail**.
2. Compute $f = \mathrm{BS2IP}(C_1)$.
3. Compute $v = \mathrm{BS2IP}(C_2)$.
4. Compute $u = v^{d_B} \bmod N_B$.
5. Compute $t = u + f \cdot 2^{l-1}$.
6. If $t \geq N_A$ then **fail**.
7. Compute $y = t^{e_A} \bmod N_A$.
8. Parse $\mathrm{I2BSP}(y, l)$ as $(w \| s)$, where $|w| = l_M + l_r$ and $|s| = l_H$.
9. Compute $c = H_2(w) \oplus s$.

10. Compute $z = G(c, l_M + l_r) \oplus w$.

11. Parse z as $(M \,||\, r)$, where $|M| = l_M$ and $|r| = l_r$.

12. If $H_1(M \,||\, r \,||\, label) = c$, then output (ACCEPT, M); otherwise output (REJECT, NULL), where ACCEPT and REJECT are symbolic values that indicate the acceptance and rejection of validity of the ciphertext, respectively, and NULL is the null string.

NOTE It is assumed that system wide parameters defined in 11.3 are implicitly known to the unsigncryption algorithm, or otherwise passed to the algorithm by such a means as the $option$. The $option$ can also be used to pass the prime factors of N_B which are required to speed up the exponentiation with the private exponent d_B in Step 4, using the Chinese Remainder Theorem (see [15], 4.3.2 Modular Arithmetic, [19] and ISO/IEC 14888-2:2008, 5.3).

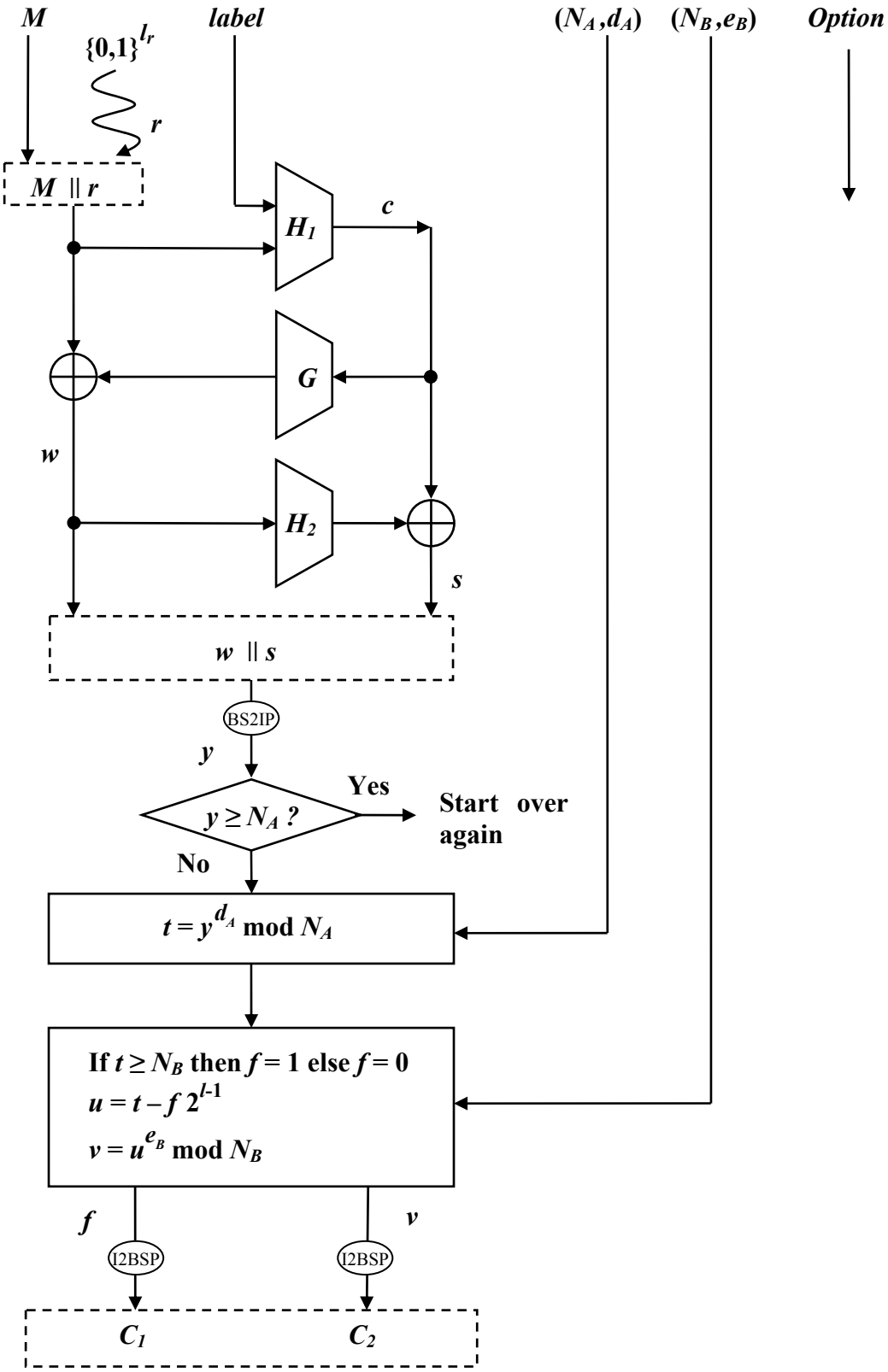

Figure 3 — Signcrypting a plaintext with *IFSC.Signcrypt*

12 Encrypt-then-sign-based mechanism (EtS)

12.1 Introduction

In this clause a signcryption mechanism based on the combination of an asymmetric encryption scheme and a digital signature scheme is defined. This mechanism is called EtS.

NOTE EtS is analysed by An, Dodis and Rabin[1]. EtS possesses proofs for confidentiality and unforgeability. The security proofs rely on the assumptions that the public-key encryption scheme is IND-CCA2 secure and that the digital signature scheme is strongly unforgeable.

12.2 Specific requirements

The EtS requires an asymmetric encryption scheme which shall be from ISO/IEC 18033-2. This asymmetric cipher will consist of three algorithms: a key generation $AC.KeyGen$, an encryption algorithm $AC.Encrypt$, and a decryption algorithm $AC.Decrypt$:

- The key generation algorithm $AC.KeyGen()$ outputs a public and private key pair (PK, pk).
- The encryption algorithm $AC.Encrypt(PK, label, M, option)$ that takes as input a public key PK, a $label$, a plaintext M, and an encryption $option$, and outputs a ciphertext C. The encryption option controls part of the execution of the encryption algorithm and the form of the ciphertext.
- The decryption algorithm $AC.Decrypt(pk, label, C)$ that takes as input a private key pk, a $label$, and a ciphertext C, and outputs a plaintext M. The decryption algorithm may **fail** under some circumstances.

The EtS requires a digital signature scheme which shall be taken from ISO/IEC 9796 or ISO/IEC 14888. This scheme will consist of three algorithms: a key generation algorithm $SS.KeyGen$, a signature generation algorithm $SS.Sign$, and a signature verification algorithm $SS.Verify$:

- The key generation algorithm $SS.KeyGen()$ outputs a public and private key pair (PK, pk).
- The signature generation algorithm $SS.Sign(pk, M)$ takes as input a private key pk and a message M, and outputs a signed message σ.
- The signature verification algorithm $SS.Verify(PK, \sigma)$ takes as input a public key PK and a signed message σ, and outputs either a message M or will **fail**.

The signcryption mechanism will inherit all the specific requirements of the asymmetric encryption scheme and digital signature scheme used in the EtS mechanism.

We also require that there exists a bit string ID_A which uniquely identifies a user A within some context.

NOTE This may be their public key value as this can be thought of as uniquely identifying a user via their public key certificate.

12.3 Key generation algorithm

Each user shall have a public and private key pair for signcryption consisting of an asymmetric encryption key pair and a digital signature key pair. This signcryption key pair may be used for both signcryption and unsigncryption. These keys are computed by the signcryption key generation algorithm as follows:

1. $(PK_e, pk_d) = AC.KeyGen()$.
2. $(PK_v, pk_s) = SS.KeyGen()$.

3. $PK = (PK_e \| PK_v)$.
4. $pk = (pk_d \| pk_s)$.
5. Output (PK, pk).

12.4 Signcryption algorithm

The signcryption algorithm $EtS.Signcrypt$ takes as input a plaintext M, a sender A's public and private key pair (PK_A, pk_A), a recipient B's public key PK_B, the sender's identifier ID_A, the recipient's identifier ID_B, a $label$, and an $option$. The sender shall perform the following steps to signcrypt the plaintext M. The resultant ciphertext is denoted by X.

1. Parse PK_A as $(PK_{A,e} \| PK_{A,v})$ and pk_A as $(pk_{A,d} \| pk_{A,s})$.
2. Parse PK_B as $(PK_{B,e} \| PK_{B,v})$.
3. Compute $C = AC.Encrypt(PK_{B,e}, M \| ID_A, label, option)$.
4. Compute $X = SS.Sign(pk_{A,s}, C \| ID_B)$.
5. Output X.

12.5 Unsigncryption algorithm

The unsigncryption algorithm $EtS.Unsigncrypt$ takes as input a ciphertext X, a sender A's public key PK_A, a recipient B's public and private key pair (PK_B, pk_B), the sender's identifier ID_A, the recipient's identifier ID_B, a $label$, and an $option$. The recipient shall perform the following steps to unsigncrypt the ciphertext X.

1. Parse PK_A as $(PK_{A,e} \| PK_{A,v})$.
2. Parse PK_B as $(PK_{B,e} \| PK_{B,v})$ and pk_B as $(pk_{B,d} \| pk_{B,s})$.
3. Compute $X' = SS.Verify(PK_{A,v}, X)$.
4. If $SS.Verify$ **fails** then **fail**.
5. Parse X' as $(C \| ID'_B)$.
6. If $ID'_B \neq ID_B$ then **fail**.
7. Compute $Y = AC.Decrypt(pk_{B,d}, label, C)$.
8. If $AC.Decrypt$ **fails** then **fail**.
9. Parse Y as $(M \| ID'_A)$.
10. If $ID'_A \neq ID_A$ then **fail**.
11. Output M.

Annex A
(normative)

Object identifiers

A.1 Formal definition

```
Signcryption {iso(1) standard(0) signcryption(29150) asn1-module(0)
signcryption-mechanisms(0) version(1)}

DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS id-sha1, id-sha256, id-sha384, id-sha512, HashFunctionAlgs FROM
DedicatedHashFunctions {iso(1) standard(0) hash-functions(10118) part(3)
asn1-module(1) dedicated-hash-functions(0)};

OID ::= OBJECT IDENTIFIER

-- Synonyms --

id-kdf OID ::= {iso(1) standard(0) encryption-algorithms(18033) part(2)
key-derivation-function(5)}

id-kdf-kdf1 OID ::= {id-kdf kdf1(1)}
id-kdf-kdf2 OID ::= {id-kdf kdf2(2)}

is29150   OID ::= {iso(1) standard(0) signcryption(29150)}
mechanism OID ::= {is29150 mechanisms(1)}

signcryption-mechanism-dlsc OID   ::= {mechanism dlsc(1)  }
signcryption-mechanism-ecdlsc OID ::= {mechanism ecdlsc(2)}
signcryption-mechanism-ifsc OID   ::= {mechanism ifsc(3)}
signcryption-mechanism-ets OID    ::= {mechanism ets(4)}

SCHashFunction ALGORITHM ::= {
{OID id-sha1 PARMS NullParms} |
{OID id-sha256 PARMS NullParms } |
{OID id-sha384 PARMS NullParms } |
{OID id-sha512 PARMS NullParms },
  ...  -- expect more hash functions here
}

SCKDFfunction ALGORITHM ::= {
{OID id-kdf-kdf1 PARMS SCHashFunction} |
{OID id-kdf-kdf2 PARMS SCHashFunction},
  ...  -- expect additional KDF functions here
}

SCparameters ::= SEQUENCE {
  kdf    SCKDFfunction,
  hash   SCHashFunction
```

```
}

SigncryptionMechanism ALGORITHM ::= {
  {OID signcryption-mechanism-dlsc   PARMS SCparameters} |
  {OID signcryption-mechanism-ecdlsc PARMS SCparameters} |
  {OID signcryption-mechanism-ifsc   PARMS SCparameters} |
  {OID signcryption-mechanism-ets    PARMS SCparameters}
}

NullParms ::= NULL

-- Cryptographic algorithm identification --

ALGORITHM ::= CLASS {
   &id   OBJECT IDENTIFIER UNIQUE,
   &Type OPTIONAL
}  WITH SYNTAX {OID &id [PARMS &Type]}


END  -- Signcryption --
```

A.2 Use of subsequent object identifiers

Each of the signcryption mechanisms specified in this International Standard uses a key derivation function. Therefore, the signcryption mechanism object identifier may include the object identifier of one of the key derivation identifiers specified in ISO/IEC 18033-2 and any associated parameters.

In addition, both mechanisms DLSC and ECDLSC employ a full domain cryptographic hash function, whereas mechanism IFSC employs two independent standard cryptographic hash functions. Since a full domain cryptographic hash function can be obtained from a standard cryptographic hash function by applying the procedure FDH1 specified in 7.2.2.2, the signcryption mechanism object identifier may further include the object identifier of one of the hash algorithm identifiers specified in ISO/IEC 10118-3[10] and any associated parameters.

Annex B
(informative)

Security considerations

B.1 Introduction

This Annex first discusses generic weaknesses of the signcryption mechanisms defined in this International Standard (Clause B.2). These generic weaknesses exist due to number theoretic assumptions and cryptographic hash functions on which the signcryption mechanisms depend for their security. It then proceeds to discuss security properties of cryptographic transformations which are used by the signcryption mechanisms (Clause B.3). This is followed by discussions on the random oracle model which is required for proving the security properties of these mechanisms (Clause B.4). Finally security properties of the signcryption mechanisms are discussed (Clauses B.5-B.8).

Security properties of a signcryption mechanism include those pertaining to confidentiality and unforgeability.

In this Annex, a *polynomial time algorithm/attacker* means one whose running time is bounded from above by a polynomial in the size of its input or a security parameter.

B.2 Generic weaknesses

Like all cryptographic mechanisms that rely for their security on computational complexity assumptions, such as the Discrete Logarithm assumption in Clause B.5, the Elliptic Curve Discrete Logarithm assumption in Clause B.6, and the Factorization assumption in Clause B.7, the signcryption mechanisms defined in this International Standard become insecure in an event when the underlying computational complexity assumptions no longer hold, due to such advancements as an algorithmic breakthrough in integer factorization or solving discrete logarithms (see for examples [16][18]), or the development of a practical quantum computer (see [20]).

In addition, all the signcryption mechanisms are subject to attacks that run in sub-exponential time (see [16][18]), although such attacks become impractical when public keys are sufficiently large.

Furthermore, as all the signcryption mechanisms use cryptographic hash functions defined in ISO/IEC 10118-2 and ISO/IEC 10118-3 (see [10]), a weakness in these allowable cryptographic hash functions may lead to the insecurity of the signcryption mechanisms.

B.3 Cryptographic transformations

This International Standard uses the following cryptographic transformations that are described in other standards:

- Standard cryptographic hash functions (ISO/IEC 10118-2 and ISO/IEC 10118-3, with a proviso that is not related to security);
- Cryptographic key derivation functions (see ISO/IEC 18033-2:2006, 6.2).

Security properties of these cryptographic transformations are addressed in those respective standards.

In addition, this International Standard defines a cryptographic transformation called a full domain cryptographic hash function in 7.2.2. The specific allowable full domain cryptographic hash function FDH1 described in 7.2.2.2 is obtained by chopping off lower order/right most bits of the output of a standard cryptographic hash function, followed by testing whether the resultant output, when being regarded as an

integer, is within a required range. The resultant output is valid only if it is within the required range. As FDH1 does not alter the original cryptographic hash function, nor does it expand the output length, FDH1 inherits all the security properties of the original cryptographic hash function.

B.4 Random oracle model

Every signcryption mechanism described in this International Standard employs some or all of the following three types of cryptographic transformations: key derivation functions, standard cryptographic hash functions and full domain cryptographic hash functions. It is assumed that these cryptographic functions all behave like a *random oracle*.

The above assumption on key derivation and cryptographic hash functions is an example of the so-called *random oracle methodology*.

B.5 Provable security of DLSC

Confidentiality

The Discrete Logarithm problem. Let g be an integer from $[1,\ldots,p-1]$ with order q modulo p and let a be chosen uniformly at random from $[1,\ldots,q-1]$. On input $g^a \bmod p$, an adversary either outputs a or declares failure. *The Discrete Logarithm assumption* states that the probability for a polynomial time adversary to successfully output a is negligibly small.

The Diffie-Hellman problem. Let g be an integer from $[1,\ldots,p-1]$ with order q modulo p and let a and b be integers chosen uniformly at random from $[1,\ldots,q-1]$. On input $g^a \bmod p$ and $g^b \bmod p$, an adversary either outputs the Diffie-Hellman key $K = g^{ab} \bmod p$ or declares failure. *The Diffie-Hellman assumption* states that the probability for a polynomial time adversary to successfully output $K = g^{ab} \bmod p$ is negligibly small.

The Gap Diffie-Hellman problem. Let g be an integer from $[1,\ldots,p-1]$ with order q modulo p and let a and b be integers chosen uniformly at random from $[1,\ldots,q-1]$. An adversary takes $g^a \bmod p$ and $g^b \bmod p$ as input and is granted access to a *Decisional Diffie-Hellman oracle* O^{DDH}. The adversary either outputs the Diffie-Hellman key $K = g^{ab} \bmod p$ or declares failure.

The Decisional Diffie-Hellman oracle O^{DDH} takes as input (h, $h^u \bmod p$, $h^v \bmod p$, z), and outputs either '1' when $z = h^{uv} \bmod p$, or '0' otherwise, where h and z are from $[1,\ldots,p-1]$ and u and v are from $[1,\ldots,q-1]$. It is permissible for h to be the same as g.

The *Gap Diffie-Hellman assumption* states that the probability for a polynomial time adversary to successfully output the Diffie-Hellman key $K = g^{ab} \bmod p$ is negligibly small.

The adaptive chosen ciphertext attacker. There are two aspects about an adaptive chosen ciphertext attacker, namely (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to break the confidentiality of messages communicated between Alice the sender and Bob the recipient. Specifically, the attacker wishes to find some (partial) information on a target ciphertext between Alice and Bob.

The attacker can record ciphertexts communicated between Alice and Bob. The attacker has access to both Alice's public key and Bob's public key. In addition the attacker is allowed to use Alice's signcryption algorithm *DLSC.Signcrypt* as a "flexible signcryption oracle" and Bob's unsigncryption algorithm *DLSC.Unsigncrypt* as a

"flexible unsigncryption oracle". With the flexible signcryption oracle, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will, including Bob the recipient's public key. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

With the flexible unsigncryption oracle, the attacker is allowed to pick, as a query to the oracle, any ciphertext (other than the target ciphertext) and any valid public key at will, including Alice the sender's public key. Upon receiving the ciphertext and the public key from the attacker, Bob dutifully performs unsigncryption on the ciphertext with his own private key and the public key from the attacker. He then returns the resultant plaintext to the attacker as an answer to the query.

It is assumed that when the public key in a query to the flexible signcryption oracle or the flexible unsigncryption oracle is not the fixed public key of Alice or that of Bob, the attacker has full access to the matching private key.

Formal security proof for confidentiality. [3] provides a mathematical proof for the fact that with DLSC, a polynomial time adaptive chosen ciphertext attacker described above, whose running time is limited by a polynomial in $L_b(p)$, has only a negligibly small chance in successfully breaking the confidentiality of messages between a sender and a recipient, under the assumptions for the random oracle model and the Gap Diffie-Hellman problem.

Unforgeability

The Gap Discrete Logarithm problem. Let g be an integer from $[1,\ldots,p-1]$ with order q modulo p and let a be an integer chosen uniformly at random from $[1,\ldots,q-1]$. An adversary takes $g^a \bmod p$ as input and is granted access to a *restricted Decisional Diffie-Hellman oracle* O^{rDDH}. The adversary either outputs the exponent a or declares failure.

The restricted Decisional Diffie-Hellman oracle O^{rDDH} takes as input (g, $g^a \bmod p$, $h^v \bmod p$, z), and outputs either '1' when $z = h^{av} \bmod p$, or '0' otherwise, where h and z are from $[1,\ldots,p-1]$ and v is from $[1,\ldots,q-1]$. It is permissible for h to be the same as g. Note that the difference between a Decisional Diffie-Hellman oracle O^{DDH} and a restricted Decisional Diffie-Hellman oracle O^{rDDH} is that the first two items in the input of O^{rDDH} are fixed to g and $g^a \bmod p$.

The *Gap Discrete Logarithm assumption* states that the probability for a polynomial time adversary to successfully output the exponent a is negligibly small.

The adaptive chosen message attacker. There are two aspects about an adaptive chosen message attacker: (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to forge a valid ciphertext from Alice the sender to a recipient of the attacker's choice.

The attacker has access to Alice's public key and is allowed to use Alice's signcryption algorithm *DLSC.Signcrypt* as a "flexible signcryption oracle". In addition, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will (other than Alice's public key). The attacker has access to the private key that matches the public key. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

Formal security proof for unforgeability. [3] provides a mathematical proof for the fact that with DLSC, a polynomial time adaptive chosen message attacker described above, whose running time is limited by a polynomial in $L_b(p)$, has only a negligibly small chance in successfully forging a valid ciphertext from Alice to

any recipient, for a new plaintext with which the attacker has not queried Alice's signcryption algorithm, under the assumptions for the random oracle model and the Gap Discrete Logarithm problem.

B.6 Provable security of ECDLSC

The Discrete Logarithm, the Diffie-Hellman, the Gap Diffie-Hellman and the Gap Discrete Logarithm problems all have their equivalent on elliptic curves.

ECDLSC has the same provable security properties with confidentiality and unforgeability as those of DLSC, under the same assumption for the random oracle model and the assumption on the elliptic curve equivalent of the Gap Diffie-Hellman and Gap Discrete Logarithm problems.

B.7 Provable security of IFSC

The Factorization problem. Let N be a composite integer. On input N, an adversary either outputs prime factors of N or declares failure. *The Factorization assumption* states that for a sufficiently large N, the probability for a polynomial time adversary to successfully output factors of N is negligibly small.

The RSA Inversion problem. Let N be an RSA modulus and e be a public exponent. Also let $C = M^e \bmod N$ be the RSA ciphertext for a randomly chosen plaintext M. On input C, an adversary either outputs M or declares failure. *The RSA (inversion) assumption* states that the probability for a polynomial time adversary to successfully output M is negligibly small.

Confidentiality

The adaptive chosen ciphertext attacker. There are two aspects about an adaptive chosen ciphertext attacker, namely (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to break the confidentiality of messages communicated between Alice the sender and Bob the recipient. Specifically, the attacker wishes to find some (partial) information on a target ciphertext between Alice and Bob.

The attacker can record ciphertexts communicated between Alice and Bob. The attacker has access to both Alice's public key and Bob's public key. In addition the attacker is allowed to use Alice's signcryption algorithm *IFSC.Signcrypt* as a "flexible signcryption oracle" and Bob's unsigncryption algorithm *IFSC.Unsigncrypt* as a "flexible unsigncryption oracle". With the flexible signcryption oracle, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will, including Bob the recipient's public key. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

With the flexible unsigncryption oracle, the attacker is allowed to pick, as a query to the oracle, any ciphertext (other than the target ciphertext) and any valid public key at will, including Alice the sender's public key. Upon receiving the ciphertext and the public key from the attacker, Bob dutifully performs unsigncryption on the ciphertext with his own private key and the public key from the attacker. He then returns the resultant plaintext to the attacker as an answer to the query.

It is assumed that when the public key in a query to the flexible signcryption oracle or the flexible unsigncryption oracle is not the fixed public key of Alice or that of Bob, the attacker has full access to the matching private key.

Formal security proof for confidentiality. [4] (see also [17]) provides a mathematical proof for the fact that with IFSC, a polynomial time adaptive chosen ciphertext attacker described above, whose running time is limited by a polynomial in l, has only a negligibly small chance in successfully breaking the confidentiality of messages between a sender and a recipient. The proof relies on the random oracle model and on the assumption that the RSA inversion problem is hard.

Unforgeability

The adaptive chosen message attacker. There are two aspects about an adaptive chosen message attacker: (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to forge a valid ciphertext from Alice the sender to a recipient of the attacker's choice.

The attacker has access to Alice's public key and both private and public keys of any recipient. Furthermore the attacker is allowed to use Alice's signcryption algorithm *IFSC.Signcrypt* as a "flexible signcryption oracle". Specifically, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

Formal security proof for unforgeability. [4] (see also [17]) provides a mathematical proof for the fact that with the IFSC, a polynomial time adaptive chosen message attacker described above, whose running time is limited by a polynomial in l, has only a negligibly small chance in successfully forging a valid ciphertext from Alice to any recipient for a new plaintext with which the attacker has not queried Alice's signcryption algorithm. The proof relies on the random oracle model and on the assumption that the RSA inversion problem is hard.

B.8 Provable security of EtS

The security of the EtS is based on the security of the underlying digital signature scheme and the underlying public-key encryption scheme. It does not require the use of the random oracle model, although the security of a specific signcryption scheme constructed in this manner may still only be proven secure in the random oracle model if either the underlying public-key encryption scheme or the underlying digital signature scheme are proven secure in the random oracle model.

The mechanism gives a secure method for combining a public-key encryption scheme and a digital signature scheme in a manner that preserves the security of both elements. More naive methods of combining public-key encryption and digital signature schemes may not give rise to a signcryption scheme with the same strict security guarantees. In certain cases, the combination of public-key encryption and digital signature schemes may lead to a scheme which is insecure even if the underlying schemes are secure.

Confidentiality

The adaptive chosen ciphertext attacker. There are two aspects about an adaptive chosen ciphertext attacker, namely (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to break the confidentiality of messages communicated between Alice the sender and Bob the recipient. Specifically, the attacker wishes to find some (partial) information on a target ciphertext between Alice and Bob.

The attacker can record ciphertexts communicated between Alice and Bob. The attacker has access to both Alice's public key and Bob's public key. In addition the attacker is allowed to use Alice's signcryption algorithm $EtS.Signcrypt$ as a "flexible signcryption oracle" and Bob's unsigncryption algorithm $EtS.Unsigncrypt$ as a "flexible unsigncryption oracle". With the flexible signcryption oracle, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will, including Bob the recipient's public key. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

With the flexible unsigncryption oracle, the attacker is allowed to pick, as a query to the oracle, any ciphertext (other than the target ciphertext) and any valid public key at will, including Alice the sender's public key. Upon receiving the ciphertext and the public key from the attacker, Bob dutifully performs unsigncryption on the ciphertext with his own private key and the public key from the attacker. He then returns the resultant plaintext to the attacker as an answer to the query.

Formal security proof for confidentiality. [1] provides a mathematical proof for the fact that with EtS, a polynomial-time adaptive chosen ciphertext attacker described above, whose running time is limited by a polynomial in $L_b(p)$, has only a negligible small chance in successfully breaking the confidentiality of messages between a sender and a recipient, under the assumption that the underlying public-key encryption scheme is secure against adaptive chosen ciphertext attacks.

Unforgeability

The adaptive chosen message attacker. There are two aspects about an adaptive chosen message attacker: (1) what the goal of the attacker is, and (2) what the attacker is allowed to do.

The goal of the attacker is to forge a valid ciphertext from Alice the sender to a recipient of the attacker's choice.

The attacker has access to Alice's public key and is allowed to use Alice's signcryption algorithm $EtS.Signcrypt$ as a "flexible signcryption oracle". In addition, the attacker is allowed to pick, as a query to the oracle, any plaintext and any valid public key at will. Upon receiving the plaintext and the public key from the attacker, Alice dutifully performs signcryption on the plaintext with her own private key and the public key from the attacker. She then returns the resultant ciphertext to the attacker as an answer to the query.

Formal security proof for unforgeability. [1] provides a mathematical proof for the fact that with EtS, a polynomial-time adaptive chosen message attacker described above, whose running time is limited by a polynomial in $L_b(p)$, has only a negligible small chance in successfully forging a valid ciphertext from Alice to any recipient, for a new plaintext with which the attacker has not queried Alice's signcryption algorithm, under the assumption that the underlying signature scheme is secure against adaptive chosen message attacks.

Annex C
(informative)

Guidance on use of the mechanisms

C.1 Introduction

The purpose of this annex is to provide guidance on the use of the signcryption mechanisms defined in this International Standard. To aid the selection of a specific signcryption mechanism, a comparison table of the first three signcryption mechanisms is provided in Clause C.2. Common requirements of all the four signcryption mechanisms are discussed in Clause C.3. Use of each mechanism requires the choice of certain parameters, and recommendations regarding sizes/lengths of these parameters are provided in Clauses C.4-C.7.

Suggested sizes for parameters in Tables C.2-C.5 in Clauses C.4-C.7 shall be considered to be the minimum sizes that are required to achieve corresponding security levels.

C.2 Selection of mechanism

All the four signcryption mechanisms specified in this International Standard, i.e. the discrete logarithm based signcryption mechanism (DLSC), the elliptic curve based signcryption mechanism (ECDLSC), the integer factorization based signcryption mechanism (IFSC), and the encrypt-then-sign based mechanism (EtS) possess provable security under appropriate assumptions (See Annex B). However, some mechanisms are more suitable than others for particular applications. When selecting a mechanism for use, facts and observations given in [3], [17], [23] and [24], and those listed below should be taken into consideration.

Table C.1 — Properties of signcryption mechanisms

Mechanism	DLSC	ECDLSC	IFSC
Dominant computation by sender	1 exponentiation	1 scalar product	2 exponentiations (1 with a public exponent; 1 with a private exponent)
Dominant computation by recipient	1 product of 2 exponentiations (1.17 exponentiations)	1 summand of 2 scalar products (1.17 scalar products)	2 exponentiations (1 with a public exponent; 1 with a private exponent)
Speed of signcryption / unsigncryption (1: fastest; 3: slowest)	2	1	3
Message expansion, Length of ciphertext (in bits)	$2l_q$, $\|M\|+2l_q$	$2l_q$, $\|M\|+2l_q$	l_r+l_H+1, $l_M+l_r+l_H+1=l+1$
Applicable to both short and long messages	Yes	Yes	Short messages only
Non-repudiation	Interactive or trusted 3rd party	Interactive or trusted 3rd party	Non-interactive
System wide public keys required	Yes	Yes	No
License possibly required	Yes	Yes	Yes

a) An algorithm attributed to E. G. Strauss that computes $g^a h^b$, the product of two exponentiations, faster than the straightforward exponentiation-then-multiplication can be found in [15], Answer to Exercise 4.6.4-36, Pages 696-697 (see also [2] for a comparison of related algorithms). The algorithm can be adapted to compute $aX + bY$, the summand of two scalar products on an elliptic curve, faster than the straightforward scalar multiplication-then-addition method [21] (see also [22] for techniques that require a smaller number of registers.)

b) When the lengths of moduli are equal, computational time for exponentiation or scalar product is determined mainly by the length of exponents. The exponents for DLSC and ECDLSC are of l_q bits in length. For IFSC a small prime such as 5, 17, or $2^{16} + 1 = 65537$ may be selected as a public exponent e, although the corresponding private exponent d is necessarily large (l bits in length). When such small public exponents are chosen, comparison of computational time between IFSC, DLSC and ECDLSC can be carried out by focusing on the length of private exponents/keys.

c) Non-repudiation settlement is said to be interactive if a repudiation dispute cannot be settled by a judge who is not fully trusted by the sender or the recipient, without invoking either an interactive or a non-interactive but inefficient, zero-knowledge proof protocol between the judge and the recipient.

C.3 Common requirements

All the mechanisms require the use of public key infrastructures for management of users' public keys. Public key infrastructures are defined in ISO/IEC 11770-1 and ISO/IEC 9594.

All the mechanisms require the selection of a standard cryptographic hash function, which is used directly in the integer factorization based signcryption, and indirectly in the other two signcryption mechanisms to build a full domain cryptographic hash function. It is recommended to use standard cryptographic hash functions specified in ISO/IEC 10118-2 and ISO/IEC 10118-3.

In addition, all the signcryption mechanisms require the use of a key derivation function. KDF1 and KDF2 defined in ISO/IEC 18033-2 are recommended.

C.4 Selecting sizes of parameters for DLSC

Sizes for the following parameters shall be considered:

l_p the length in bits of prime p; that is $l_p = L_b(p) = \lceil \log_2(p+1) \rceil$.

l_q the length in bits of prime q; that is $l_q = L_b(q) = \lceil \log_2(q+1) \rceil$.

Table C.2 — Parameter sizes for DLSC

Security level (in bits)	80	112	128	192	256
l_p	1 024	2 048	3 072	7 680	15 360
l_q	160	224	256	384	512

NOTE Table C.2 is based on ISO/IEC 14888-3:2006, 5.1.3.1.

C.5 Selecting sizes of parameters for ECDLSC

It is recommended that either a prime field $GF(p)$, where p is a prime, or a binary field $GF(2^m)$, where m is a positive integer, be selected to serve as the underlying finite field for an elliptic curve.

When a prime field $GF(p)$ is employed, sizes for the following parameters shall be considered:

l_p the size in bits of prime p when the underlying finite field is a prime field $GF(p)$; that is $l_p = L_b(p) = \lceil \log_2(p+1) \rceil$.

l_q the length in bits of prime q; that is $l_q = L_b(q) = \lceil \log_2(q+1) \rceil$.

Table C.3 — Parameter sizes for ECDLSC over a prime field $GF(p)$

Security level (in bits)	80	112	128	192	256
l_p	192	224	256	384	521
l_q	160	224	256	384	512

When a binary field $GF(2^m)$ is employed, sizes for the following parameters shall be considered:

m a positive integer indicating the size of the underlying binary field $GF(2^m)$.

l_q the length in bits of prime q; that is $l_q = L_b(q) = \lceil \log_2(q+1) \rceil$.

Table C.4 — Parameter sizes for ECDLSC over a binary field $GF(2^m)$

Security level (in bits)	80	112	128	192	256
m	163	233	283	409	571
l_q	160	224	256	384	512

NOTE Tables C.3 and C.4 are based on FIPS 186-3, Appendix D (See [5]). Examples of elliptic curves in the same document are recommended.

C.6 Selecting sizes of parameters for IFSC

Sizes for the following parameters shall be considered:

l the length in bits of an RSA modulus.

l_r the length in bits of a random number r generated in the first step of the signcryption algorithm.

l_H the length in bits of the output of a standard cryptographic hash function H.

Table C.5 — Parameter sizes for IFSC

Security level (in bits)	80	112	128	192	256
l	1 024	2 048	3 072	7 680	15 360
l_r	80	112	128	192	256
l_H	160	224	256	384	512

NOTE 1 $l = l_M + l_r + l_H$, where l_M is the length in bits of plaintexts.

NOTE 2 Parameter l in Table C.5 is chosen to be identical to parameter l_p in Table C.2, based on the assumption that factorization and discrete logarithm problems have a similar degree of hardness.

C.7 Selecting sizes of parameters for EtS

The security of EtS is based on the security of the underlying public-key encryption scheme and signature scheme. The construction is secure if it is instantiated with a secure public-key encryption scheme and a secure signature scheme. Parameter sizes should be selected for the encryption and signature key pairs in such a way that security is guarantee for these schemes. The reader should refer to literature on selecting parameter sizes for the underlying schemes when selecting parameter sizes for this signcryption scheme.

Annex D
(informative)

Examples

D.1 Introduction

This Annex contains worked examples of the operation of the four signcryption mechanisms specified in this International Standard, these being the discrete logarithm based signcryption mechanism (DLSC), the elliptic curve based signcryption mechanism (ECDLSC), and the integer factorization based signcryption mechanism (IFSC) and the encrypt-then-sign based mechanism (EtS). In the examples, integers and bit strings in typewriter font are expressed in hexadecimal notation, unless otherwise specified. Integers are right-justified in an appropriate field, and bit strings are left-justified so that their lengths in bits are a multiple of four. Spaces and line breaks are inserted for better readability.

D.2 Example for DLSC

System wide parameters

DLSC has seven system wide parameters: l_q, l_p, q, p, g, G and H. This example assumes that $l_q = 224$ and $l_p = 2048$. In addition, G is instantiated by the key derivation function KDF2 (see 7.3) with a 32-bit counter starting at 1, and H by the full domain hash function FDH1 (see 7.2.2.2). Furthermore, SHA-224 as specified in Amendment 1 to ISO/IEC 10118-3:2004 assumes the role of the underlying cryptographic hash function for both KDF2 and FDH1. The values of the remaining three system wide parameters q, p and g are represented below in hexadecimal notation.

q = 801C0D34 C58D93FE 99717710 1F80535A 4738CEBC BF389A99
B36371EB

p = AD107E1E 9123A9D0 D660FAA7 9559C51F A20D64E5 683B9FD1
B54B1597 B61D0A75 E6FA141D F95A56DB AF9A3C40 7BA1DF15
EB3D688A 309C180E 1DE6B85A 1274A0A6 6D3F8152 AD6AC212
9037C9ED EFDA4DF8 D91E8FEF 55B7394B 7AD5B7D0 B6C12207
C9F98D11 ED34DBF6 C6BA0B2C 8BBC27BE 6A00E0A0 B9C49708
B3BF8A31 70918836 81286130 BC8985DB 1602E714 415D9330
278273C7 DE31EFDC 7310F712 1FD5A074 15987D9A DC0A486D
CDF93ACC 44328387 315D75E1 98C641A4 80CD86A1 B9E587E8
BE60E69C C928B2B9 C52172E4 13042E9B 23F10B0E 16E79763
C9B53DCF 4BA80A29 E3FB73C1 6B8E75B9 7EF363E2 FFA31F71
CF9DE538 4E71B81C 0AC4DFFE 0C10E64F

g = AC4032EF 4F2D9AE3 9DF30B5C 8FFDAC50 6CDEBE7B 89998CAF
74866A08 CFE4FFE3 A6824A4E 10B9A6F0 DD921F01 A70C4AFA
AB739D77 00C29F52 C57DB17C 620A8652 BE5E9001 A8D66AD7
C1766910 1999024A F4D02727 5AC1348B B8A762D0 521BC98A
E2471504 22EA1ED4 09939D54 DA7460CD B5F6C6B2 50717CBE
F180EB34 118E98D1 19529A45 D6F83456 6E3025E3 16A330EF
BB77A86F 0C1AB15B 051AE3D4 28C8F8AC B70A8137 150B8EEB
10E183ED D19963DD D9E263E4 770589EF 6AA21E7F 5F2FF381
B539CCE3 409D13CD 566AFBB4 8D6C0191 81E1BCFE 94B30269
EDFE72FE 9B6AA4BD 7B5A0F1C 71CFFF4C 19C418E1 F6EC0179
81BC087F 2A7065B3 84B890D3 191F2BFA

Public and private key pairs

The sender's public and private key pair (y_A, x_A) and the recipient's public and private key pair (y_B, x_B) are assigned the following values in hexadecimal notation.

y_A = 0C3618A2 2B37A58E EC74D9B7 AAFA24E9 E2A32415 FD13BF19
91D79864 48E8CF56 23B074C0 5ED99F85 AC75D0DE 7FF4131D
7854210C 2C88FB41 6211D6E8 0D37ECF1 5F2A8536 8F8953F7
03FCCFD1 972ED722 0A0AA9FD A1AC3FC0 85641081 D07B2C0C
93226456 EB9FB8DE BBA46080 8806AA70 E4543604 8898C7ED
0FCA25EF 4D796D18 D80D97AA 89C1DB49 371FF49C 2BB1FBA4
CCC0690D BD2E66AC FCC9C1D4 DA8F5CCC C5795C23 8160F903
815FA73C 1D84459D 5380D508 02815E74 29A411D3 5899B97D
DE4D800B FF7C84FF 2E7796BD 23B61DF9 FD4F1CCE A84CF110
EBD03FF2 454851E1 E4B0C2C3 D360D427 63C5B48A E2132DFB
BC19AB95 E1FC2331 B792C6DD 5A71B120

x_A = 51941AE6 01418614 86A16DEE 6B133EA7 2513AF12 462D7A49
0A53F4BB

y_B = 1E94AEF7 E5E5AACA E51E3C8B BA22F33D 27DA5939 A616F7C0
19CCC3ED 295EF62B 230E26A3 B21AA147 1413B4D8 4D46E779
B5450067 C16C6783 20B5908D 2B47A35C 7EAC9DB6 CC550F5F
6DAC06EB BF949CFE 1C1C2551 BB4AE046 E9FB4CAB 0B462C9A
A7F0E88B 3E03F1BB F9DF8877 22300F75 14F6E0D1 9868E55A
2A25D2FE 0C2E4B35 42188587 D9AE4A0C 1A618760 BA7D6DB1
4DE2CC2C 7614FC98 4A2EE754 0A3FBEF8 CA794125 578D4843
E319B0DB 84763D06 4B0ABD44 C4D84202 31DF3F96 8686B702
C0BDF981 93E3D8DE 0E95DCA7 B610D66E 9B5296C7 AF84ED3F
FB437158 BCF499B6 C624820B 858006FB E32E7F8E E0DDA1B7
3A406113 D071722D AD469C26 7B1C9599

x_B = 11FB849F 5EF90A5A DA6444DC A522EDA2 8C4D4ADA 81D378C7
5B0458EA

Message and label

The message M is '`This is the message to be signcrypted`' and the encryption label is '`0001`', without the quotation marks. Their hexadecimal values are as follows.

M = 54686973 20697320 74686520 6D657373 61676520 746F2062
65207369 676E6372 79707465 64

label = 30303031

Values in signcryption process

The following sequence of values in hexadecimal notation trace key intermediate variables leading to the resultant ciphertext X when the signcryption algorithm *DLSC.Signcrypt* is applied to the message M.

u = 595970EF CBAFB118 EB0E8171 816E8901 671E3E11 CF9BD976
609FDDA9

K = A11EDC14 3F0C2E2F D0495BD2 BCEB6DF4 5655FB91 EA0F06A2
6786B100 E1ED1A58 BD5BCCD2 60D34114 8701F21B 0373D302

903E4193 89ED4858 D38FDB93 11607F8A 44CBA98B 32FDA4EC
2F93AF19 4D50AB37 59AA7E7D 3204EBFE BB61CFAD 21356C1F
928911F6 7C1FD9E9 1D79AFC0 772BFC82 481386BF DCD599D5
17C88050 D5461F37 AD43A9BB 8F012A21 860B78CB 86ED0591
49226F08 A5D98458 E4538453 DBE5CD63 865543DA BF9CA670
6810EB83 8871206E 84166DCD 880F3A45 3035D818 02DF02AD
D4B60E08 C37098EA EA99D886 A0698CDB 0E22B3A7 16E68FAD
A1D565EE F8BFF693 85E8C4BF A115ED9E DA7E69DC B9AF35F1
B82B86AF BACA5BFC F0EDEC1F DFA0AD76

$G(\mathrm{I2BSP}(K, l_p), |M|)$ =

0D243616 72903B4E 2F6B9917 E9DA0086 D3BAF174 00AEF710
720EB31C DF534B7F 92616FE3 CA

C = 594C5F65 52F9486E 5B03FC37 84BF73F5 B2DD9454 74C1D772
172EC075 B83D280D EB111B86 AE

Input to H =

A11EDC14 3F0C2E2F D0495BD2 BCEB6DF4 5655FB91 EA0F06A2
6786B100 E1ED1A58 BD5BCCD2 60D34114 8701F21B 0373D302
903E4193 89ED4858 D38FDB93 11607F8A 44CBA98B 32FDA4EC
2F93AF19 4D50AB37 59AA7E7D 3204EBFE BB61CFAD 21356C1F
928911F6 7C1FD9E9 1D79AFC0 772BFC82 481386BF DCD599D5
17C88050 D5461F37 AD43A9BB 8F012A21 860B78CB 86ED0591
49226F08 A5D98458 E4538453 DBE5CD63 865543DA BF9CA670
6810EB83 8871206E 84166DCD 880F3A45 3035D818 02DF02AD
D4B60E08 C37098EA EA99D886 A0698CDB 0E22B3A7 16E68FAD
A1D565EE F8BFF693 85E8C4BF A115ED9E DA7E69DC B9AF35F1
B82B86AF BACA5BFC F0EDEC1F DFA0AD76 54686973 20697320
74686520 6D657373 61676520 746F2062 65207369 676E6372
79707465 640C3618 A22B37A5 8EEC74D9 B7AAFA24 E9E2A324
15FD13BF 1991D798 6448E8CF 5623B074 C05ED99F 85AC75D0
DE7FF413 1D785421 0C2C88FB 416211D6 E80D37EC F15F2A85
368F8953 F703FCCF D1972ED7 220A0AA9 FDA1AC3F C0856410
81D07B2C 0C932264 56EB9FB8 DEBBA460 808806AA 70E45436
048898C7 ED0FCA25 EF4D796D 18D80D97 AA89C1DB 49371FF4
9C2BB1FB A4CCC069 0DBD2E66 ACFCC9C1 D4DA8F5C CCC5795C
238160F9 03815FA7 3C1D8445 9D5380D5 0802815E 7429A411
D35899B9 7DDE4D80 0BFF7C84 FF2E7796 BD23B61D F9FD4F1C
CEA84CF1 10EBD03F F2454851 E1E4B0C2 C3D360D4 2763C5B4
8AE2132D FBBC19AB 95E1FC23 31B792C6 DD5A71B1 201E94AE
F7E5E5AA CAE51E3C 8BBA22F3 3D27DA59 39A616F7 C019CCC3
ED295EF6 2B230E26 A3B21AA1 471413B4 D84D46E7 79B54500
67C16C67 8320B590 8D2B47A3 5C7EAC9D B6CC550F 5F6DAC06
EBBF949C FE1C1C25 51BB4AE0 46E9FB4C AB0B462C 9AA7F0E8
8B3E03F1 BBF9DF88 7722300F 7514F6E0 D19868E5 5A2A25D2
FE0C2E4B 35421885 87D9AE4A 0C1A6187 60BA7D6D B14DE2CC
2C7614FC 984A2EE7 540A3FBE F8CA7941 25578D48 43E319B0
DB84763D 064B0ABD 44C4D842 0231DF3F 968686B7 02C0BDF9
8193E3D8 DE0E95DC A7B610D6 6E9B5296 C7AF84ED 3FFB4371
58BCF499 B6C62482 0B858006 FBE32E7F 8EE0DDA1 B73A4061
13D07172 2DAD469C 267B1C95 99303030 31

r = 1C69A661 68A5BD22 52E74F28 2D9E3884 40324D71 558CE87E
26E9BE9C

s = 3D8F7289 A79357A2 30806908 EAC4BCEB 8748889C B58C93FC
BC2FA0E1

X = 594C5F65 52F9486E 5B03FC37 84BF73F5 B2DD9454 74C1D772

172EC075 B83D280D EB111B86 AE1C69A6 6168A5BD 2252E74F
282D9E38 8440324D 71558CE8 7E26E9BE 9C3D8F72 89A79357
A2308069 08EAC4BC EB874888 9CB58C93 FCBC2FA0 E1

D.3 Example for ECDLSC

System wide parameters

This example is for ECDLSC over a prime field $GF(p)$. The example requires nine system wide parameters: l_p, p, V, l_q, q, J, fmt, G and H. Specifically it is assumed that $l_q = l_p = 256$, $fmt = uncompressed$, G is instantiated by the key derivation function KDF2 (see 7.3) with a 32-bit counter starting at 1, and H by the full domain hash function FDH1 (see 7.2.2.2). Furthermore, SHA-256 as specified in ISO/IEC 10118-3 is assumed to be the underlying cryptographic hash function for both KDF2 and FDH1. The remaining four system wide parameters p, V, J and q are represented below in hexadecimal notation.

p = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFF

The elliptic curve V is specified by two elements a and b in $GF(p)$ satisfying an equation of the form $y^2 = x^3 + ax + b \pmod{p}$.

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC

b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6
3BCE3C3E 27D2604B

The designated base point J on the elliptic curve V is specified by a pair of coordinates J_x and J_y, i.e. $J = (J_x, J_y)$.

J_x = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0
F4A13945 D898C296
J_y = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE
CBB64068 37BF51F5

q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84
F3B9CAC2 FC632551

Public and private key pairs

The sender's public and private key pair (Y_A, x_A) and the recipient's public and private key pair (Y_B, x_B) are assigned the following values in hexadecimal notation, where Y_A is specified by a pair of coordinates $Y_{A,x}$ and $Y_{A,y}$, and Y_B by a pair of coordinates $Y_{B,x}$ and $Y_{B,y}$.

$Y_A = (Y_{A,x}, Y_{A,y})$
$Y_{A,x}$ = AE179A2F 2267BC7F D9FF31B6 D598D690 568A3558 2A90FA10
C8B471DB 8B7E96E1
$Y_{A,y}$ = 9127ED65 DF0B18EC 3E7760F2 448D9B54 89DDB111 2FD141E1
DCEA9C44 6017CFB6

x_A = D81EE599 206D7498 719218D6 9E7B0060 B00FF644 59253CEC
D17F590E 94599527

$Y_B = (Y_{B,x}, Y_{B,y})$

$Y_{B,x}$ = 78903C9E 5DE72BCF 1E7868EB 9B5DE376 435CC195 53A80DC6
83ED571D 08D44B9A

$Y_{B,y}$ = 5FD52F5E E16A2037 45BF3AC7 ABBB1905 B2626CA8 21AADAB1
8CAB0498 19182A67

x_B = 7A30CD41 0B0BEEC9 B68E7056 E64ED14C 083DCB7A 89559EFD
290A129E 76D4DE1D

Message and label

The message M is '`This is the message to be signcrypted`' and the encryption label is '`0002`', without the quotation marks. Their hexadecimal values are as follows.

M = 54686973 20697320 74686520 6D657373 61676520 746F2062
65207369 676E6372 79707465 64

label = 30303032

Values in signcryption process

The following sequence of values in hexadecimal notation trace key intermediate variables leading to the resultant ciphertext X when the signcryption algorithm *ECDLSC.Signcrypt* is applied to the message M.

u = 709A1E5C 456C7737 92EDD968 ABCE4F39 6DFCD32D 4136C122
07F6452E 6AA60190

$K = (K_x, K_y)$

K_x = 5564C315 DDE7FD0E F8269ACC 27DB4292 A0D8027C BFA91CCD
CBF8FE71 A7252995

K_y = 7F3C6EEF 61DD1C11 E2AFA901 57C15DDA 0EE2BB82 3AA4745B
33E11FC4 9608B4AA

$G(\mathrm{I2BSP}(K, l_p), |M|)$ =
ADEFD20D E2522A8E 032D7DC8 35B2715D A55183BA 81E6FC89
D0C5EA0F 7F25A2C4 2A0FF3FD 8C

C = F987BB7E C23B59AE 774518E8 58D7022E C436E69A F589DCEB
B5E59966 184BC1B6 537F8798 E8

Input to H =
8AAC9862 BBBCFFA1 DF04D359 84FB6852 541B004F 97F52399
B97F1FCE 34E4A532 AFE78DDD EC3BA382 3C55F520 2AF82BBB
41DC5770 47548E8B 667C23F8 92C11695 4A8D0D2E 640D2E64
0E8D0CA4 0DACAE6E 6C2CECA4 0E8DE40C 4CA40E6D 2CEDCC6E
4F2E0E8C AC92B85E 68BC899E F1FF67FC C6DB5663 5A415A28
D560AA43 E84322D1 C76E2DFA 5B86449F B5977C2C 63B0F9DD
83C91236 6D522776 C444BF45 078773AA 7111805F 3EDA3C48
1E4F2EF3 95E78F3C 3475CDAE F1BB21AE 60CAA9D4 06E341F6

```
        AB8E846A 25CD2FEA 97AF70B5 101BA2DF 9D63D5DD 8C82D931
        365410D5 6D58C655 824C0C8C 15339818 18190

r =     38E1D6E5 563B649C F1B3B626 D39A9844 9CA3AA51 A5EEBBFD
        07DD6030 C6B72672

s =     29C51FCB 1987AC7B 55942920 CEB69A99 C97A549B 8858999E
        87A8DB58 92D26754

X =     F987BB7E C23B59AE 774518E8 58D7022E C436E69A F589DCEB
        B5E59966 184BC1B6 537F8798 E838E1D6 E5563B64 9CF1B3B6
        26D39A98 449CA3AA 51A5EEBB FD07DD60 30C6B726 7229C51F
        CB1987AC 7B559429 20CEB69A 99C97A54 9B885899 9E87A8DB
        5892D267 54
```

NOTE In order to represent the input to *H* in a hexadecimal notation, its bit string is left-justified with three extra '0' bits being appended to the right.

D.4 Example for IFSC

System wide parameters

IFSC requires seven (7) system wide parameters: l_r, l_H, l_M, l, G, H_1 and H_2. Values for the first four parameters are assumed to be $l_r = 80$, $l_H = 160$, $l_M = 784$ and $l = 1024$. G is instantiated by the key derivation function KDF2 (see 7.3) with a 32-bit counter starting at 1 and SHA-1 assuming the role of the underlying cryptographic hash function. Both H_1 and H_2 are instantiated by the full domain hash function FDH1 (see 7.2.2.2), with SHA-1 being the underlying cryptographic hash function for H_1 and SHA-256 whose output is truncated to the first/leftmost 160 bits being the underlying cryptographic hash function for H_2 respectively. Both SHA-1 and SHA-256 are specified in ISO/IEC 10118-3.

Public and private key pairs

In the example $2^{16} + 1 = 65537$ is adopted as the public exponent for both the sender and the recipient. The sender's public and private key pair $[(N_A, e_A), (N_A, d_A)]$ and the recipient's public and private key pair $[(N_B, e_B), (N_B, d_B)]$ are specified below in hexadecimal notation. N_A and N_B are preceded by their 512-bit prime factors p_A and q_A, and p_B and q_B respectively.

```
p_A =   F49C89BB 4D44251A 5B3582AA C833632A 175C9702 1E84CE1E
        138AB83F 37257277 D1B575EC AC7C7EA3 D7EAF4B0 83BEF54F
        D157E97A 8EAE77F0 2B50551A A2CC7CB5

q_A =   BEC2632C 2F9C5745 0B1BC2E3 8352530D 1D18E74C 1CF6E0B5
        B8294710 13BD0D22 98B01CF5 1E6EBD77 61F6A3D6 ED17589F
        6FF88B6E 82AB95FC C6B5D7E3 87BC946F

N_A =   p_A × q_A =
        B645EB9C 24F25C50 CCAA070F 2B40A168 DD411882 747D4014
        978041DD 44C346C2 F240A642 03A61085 6700B5C5 5FB73E94
        AF3B40F2 7BC14DC9 F1D48553 0B49677C 5C913876 6999ACFC
        49C48652 48A2E087 09E805C0 0C317AF7 DA51125F 9E0F1C47
        BBE9CF68 17383E16 EEF366D5 90E70ADF 531469A0 986F3999
        06794818 D6AEB67B
```

e_A = 10001 (i.e. 65537 in decimal)

d_A = 1B59BE92 FD90487D EF580C1B 20986020 06234074 42DA9085
067EBE63 5F7015EC 26C1C0B0 E32F4607 5E7F62C7 2D27A802
5391069B DC2A136C 1C1791B1 F65D9B63 72841386 6CCB151A
559EF3B6 F5022D59 31643022 C9F89D9B 437E1A60 36BAD8FF
C0DDA0DF 5E0FA81E 86FB1703 EE17D4E2 0928684E 29BCE982
B6C6D6E8 57B08671

p_B = F6141760 AB134B9B EF88A1CE 067F5628 8CA6A863 08D0A234
D6B5B9F6 B1D6570B F66F6AB5 1D8CB20B 20F78371 C770BA9D
775202F2 3BBF97CE 5D71E3B0 7002AF5B

q_B = C5AE5457 49FE9374 7D75DCA5 7C966201 060F2DC8 C22A97F0
19E5593A FCD3FA54 099BFF0B 54A71827 A1DB4F98 6F0AFB7C
F0565284 4478ACE4 BD8DE93C 4B06962B

N_B = $p_B \times q_B$ =
BE0508B7 C6FA3AA0 21814268 10407544 C76E7C1E A47A929F
9C0C7002 8FC83DE9 F8E0E455 814BDF0B 25643883 FFC0CF26
4DFEBF9F B95BB8F6 77F4F39A E11CFF21 4A16FF69 4937FA7D
BE8CDC4F F97A9754 165429F5 8029F128 DDA968B9 8D45F652
0DAF5C6F EC09F56C FFFC99A9 CC451593 A866070F 7901EA13
EAD7F608 2854C649

e_B = 10001 (i.e. 65537 in decimal)

d_B = 24E99F6E EBE5521C 27460FDB 5D44D842 FB26D84F AF8DEC0B
BB69A31C AC47AED7 53B48446 A3EE542E BB1DC3E8 C876F106
B4E206E1 85456F21 D9DDA8B0 EE880E7B 57CFD03D 0366BFAF
35C79D2E 2585A4E0 825EF051 DFB36A74 05C9E20D 99CDCC76
04F6533F 89407477 44C867C0 3A7DBAE1 38156179 A0E2B310
A26F9054 B69F932D

Message and label

The message M is a bit string '01010101...01' of length 784, or equivalently, a string of 98 repetitions of the letter 'U', and the encryption label is '0003', without the quotation marks. Their values in hexadecimal notation are as follows.

M = 55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
5555

label = 30303033

Values in signcryption process

The following sequence of values trace key intermediate variables leading to the resultant ciphertext X when the signcryption algorithm *IFSC.Signcrypt* is applied to the message M. All the values, except that of f, are in hexadecimal notation. The value of f is a single binary bit.

r = 257753B8 A72F7759 526F

Input to H_1 =

```
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55552577 53B8A72F 7759526F 30303033
```

c = 6B8BE588 B0468FC9 E17740FC 729CA3F9 3FED0A04

$G(c, l_M + l_r)$ =

```
ACB3DDE6 A7A92647 274817FF 1BA0116F 89583DE8 D36F7E5D
0D525C36 E8C21F89 A1BF40E1 2A18A320 E06FC96F AE69B36A
77DC922D 3CD636F3 0AAC7ACC ACEF6833 A000F90C 8B3DE619
BF294F3C 3A6ED994 13342124 FD9A191B 3FC5664D 28FD890B
93659CD6 2FDB62E1 14F9E391
```

w =

```
F9E688B3 F2FC7312 721D42AA 4EF5443A DC0D68BD 863A2B08
58070963 BD974ADC F4EA15B4 7F4DF675 B53A9C3A FB3CE63F
2289C778 698363A6 5FF92F99 F9BA3D66 F555AC59 DE68B34C
EA7C1A69 6F3B8CC1 46617471 A8CF4C4E 6A903318 7DA8DC5E
C630B9A1 7C63C5CE 63A0B1FE
```

$H_2(w)$ = 85C2229F D11875F1 5C0FA519 99F18E9A 78697623

s = EE49C717 615EFA38 BD78E5E5 EB6D2D63 47847C27

$\mathrm{BS2IP}(w \| s)$ =

```
F9E688B3 F2FC7312 721D42AA 4EF5443A DC0D68BD 863A2B08
58070963 BD974ADC F4EA15B4 7F4DF675 B53A9C3A FB3CE63F
2289C778 698363A6 5FF92F99 F9BA3D66 F555AC59 DE68B34C
EA7C1A69 6F3B8CC1 46617471 A8CF4C4E 6A903318 7DA8DC5E
C630B9A1 7C63C5CE 63A0B1FE EE49C717 615EFA38 BD78E5E5
EB6D2D63 47847C27
```

Since $\mathrm{BS2IP}(w \| s) \geq N_A$, start over to choose a new random r.

r = 5BFA4BDB 99DC5246 9625

Input to H_1 =

```
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555555 55555555 55555555 55555555 55555555 55555555
55555BFA 4BDB99DC 52469625 30303033
```

c = 7EC4FC35 92D0C05F D4E3F364 D850580D E6F99DAB

$G(c, l_M + l_r)$ =

```
43971C20 F2E0CB0D F88FF6C5 4465C8E1 616666A4 8EC8E5A5
A57A160F 3246AFE4 1F518631 D7FD2B62 24D2435E EE6951B2
7F94D4C0 63D96F39 C68C4BE4 432B699F 56A62C98 C32DD766
DA36B0E2 66301DE5 A86C123E 7E772C54 635B4F30 0CCC1E51
A9B24944 12F29C43 C3BB3D63
```

w =

```
16C24975 A7B59E58 ADDAA390 11309DB4 343333F1 DB9DB0F0
F02F435A 6713FAB1 4A04D364 82A87E37 7187160B BB3C04E7
2AC18195 368C3A6C 93D91EB1 167E3CCA 03F379CD 96788233
8F63E5B7 336548B0 FD39476B 2B227901 360E1A65 59994B04
```

FCE712BE 5929059F 91FDAB46

$H_2(w)$ = 28E9066D EB817636 BB4D5FA9 11D54CEF 146B4596

s = 562DFA58 7951B669 6FAEACCD C98514E2 F292D83D

$\mathrm{BS2IP}(w \| s)$ =
16C24975 A7B59E58 ADDAA390 11309DB4 343333F1 DB9DB0F0
F02F435A 6713FAB1 4A04D364 82A87E37 7187160B BB3C04E7
2AC18195 368C3A6C 93D91EB1 167E3CCA 03F379CD 96788233
8F63E5B7 336548B0 FD39476B 2B227901 360E1A65 59994B04
FCE712BE 5929059F 91FDAB46 562DFA58 7951B669 6FAEACCD
C98514E2 F292D83D

t = 981B81A6 D52F6718 DD0669E1 71F4E406 0BCE0049 DA1BCFC7
8996E66D ED73CEBF 1455B8C4 FAB4F8D9 392D5093 2B25A47E
4FCF3FEA DD39E48B 1F4BD7B4 491C243E B3B4E3DE 6B14EB57
24576F1C A7999778 A7A8E88A 3D9F3357 6BA3E7BF 1E25D20D
FD79C327 E5F887E0 A33BCFD1 685D89E1 B73349A2 29F77DE0
9B8B7E03 C2C9AF73

f = 0

v = 77591B23 9D2772B6 4519BF4E A6407E7E D657088E BF17E1C0
E7B1DEB5 A4C8889E 1FAC5A57 16BCFC4B 1D064434 71174D7C
307020FD 97221B5D D2A6E0A7 820CDEA4 60ADBDB9 905B737F
C97233BD 77285D72 6DA53A3B 13A056BD 4960C264 1687C099
6B5B9DC9 7E7E2FD4 D6856873 175C4499 1F39F139 486BBCC9
29DA3D96 BE4E3BB4

X = 3BAC8D91 CE93B95B 228CDFA7 53203F3F 6B2B8447 5F8BF0E0
73D8EF5A D264444F 0FD62D2B 8B5E7E25 8E83221A 388BA6BE
1838107E CB910DAE E9537053 C1066F52 3056DEDC C82DB9BF
E4B919DE BB942EB9 36D29D1D 89D02B5E A4B06132 0B43E04C
B5ADCEE4 BF3F17EA 6B42B439 8BAE224C 8F9CF89C A435DE64
94ED1ECB 5F271DDA 0

NOTE 1 f is a binary bit.

NOTE 2 In order to represent the resultant ciphertext X in hexadecimal notation which requires that the total number of bits in X be a multiple of four (4), X is left-justified with three extra '0' bits being appended to the right. As a result, the right most hexadecimal character is either '0' or '8'.

D.5 Example for EtS

This is a worked example for EtS. SHA-1, KDF2 and RSAES are used for public key encryption as in ISO/IEC 18033-2. SHA-1 and RSA-PSS are used for signature generation as in ISO/IEC 14888-2.

In the example, $2^{16}+1 = 65537$ is adopted as the public exponent for both the sender and the recipient. The signature verification and generation portion of a sender A's public and private key pair is denoted by $[(N_A, e_A), (N_A, d_A)]$. The asymmetric encryption and decryption portion of a recipient B's public and private key pair is denoted by $[(N_B, e_B), (N_B, d_B)]$. The recipient B's public key (N_B, e_B) is used in the encryption part and the sender A's private key (N_A, d_A) is used in the signature part.

These keys are specified below in hexadecimal notation. N_A and N_B are preceded by their 512-bit prime factors p_A and q_A, and p_B and q_B respectively.

p_A = F49C89BB 4D44251A 5B3582AA C833632A 175C9702 1E84CE1E
138AB83F 37257277 D1B575EC AC7C7EA3 D7EAF4B0 83BEF54F
D157E97A 8EAE77F0 2B50551A A2CC7CB5

q_A = BEC2632C 2F9C5745 0B1BC2E3 8352530D 1D18E74C 1CF6E0B5
B8294710 13BD0D22 98B01CF5 1E6EBD77 61F6A3D6 ED17589F
6FF88B6E 82AB95FC C6B5D7E3 87BC946F

N_A = $p_A \times q_A$ =
B645EB9C 24F25C50 CCAA070F 2B40A168 DD411882 747D4014
978041DD 44C346C2 F240A642 03A61085 6700B5C5 5FB73E94
AF3B40F2 7BC14DC9 F1D48553 0B49677C 5C913876 6999ACFC
49C48652 48A2E087 09E805C0 0C317AF7 DA51125F 9E0F1C47
BBE9CF68 17383E16 EEF366D5 90E70ADF 531469A0 986F3999
06794818 D6AEB67B

e_A = 10001 (i.e. 65537 in decimal)

d_A = 1B59BE92 FD90487D EF580C1B 20986020 06234074 42DA9085
067EBE63 5F7015EC 26C1C0B0 E32F4607 5E7F62C7 2D27A802
5391069B DC2A136C 1C1791B1 F65D9B63 72841386 6CCB151A
559EF3B6 F5022D59 31643022 C9F89D9B 437E1A60 36BAD8FF
C0DDA0DF 5E0FA81E 86FB1703 EE17D4E2 0928684E 29BCE982
B6C6D6E8 57B08671

p_B = F6141760 AB134B9B EF88A1CE 067F5628 8CA6A863 08D0A234
D6B5B9F6 B1D6570B F66F6AB5 1D8CB20B 20F78371 C770BA9D
775202F2 3BBF97CE 5D71E3B0 7002AF5B

q_B = C5AE5457 49FE9374 7D75DCA5 7C966201 060F2DC8 C22A97F0
19E5593A FCD3FA54 099BFF0B 54A71827 A1DB4F98 6F0AFB7C
F0565284 4478ACE4 BD8DE93C 4B06962B

N_B = $p_B \times q_B$ =
BE0508B7 C6FA3AA0 21814268 10407544 C76E7C1E A47A929F
9C0C7002 8FC83DE9 F8E0E455 814BDF0B 25643883 FFC0CF26
4DFEBF9F B95BB8F6 77F4F39A E11CFF21 4A16FF69 4937FA7D
BE8CDC4F F97A9754 165429F5 8029F128 DDA968B9 8D45F652
0DAF5C6F EC09F56C FFFC99A9 CC451593 A866070F 7901EA13
EAD7F608 2854C649

e_B = 10001 (i.e. 65537 in decimal)

d_B = 24E99F6E EBE5521C 27460FDB 5D44D842 FB26D84F AF8DEC0B
BB69A31C AC47AED7 53B48446 A3EE542E BB1DC3E8 C876F106
B4E206E1 85456F21 D9DDA8B0 EE880E7B 57CFD03D 0366BFAF
35C79D2E 2585A4E0 825EF051 DFB36A74 05C9E20D 99CDCC76
04F6533F 89407477 44C867C0 3A7DBAE1 38156179 A0E2B310
A26F9054 B69F932D

ID_A = 00003141

ID_B = FFFF0097

M = 55555555 55555555 55555555 55555555 55555555

$M||ID_A$ = 55555555 55555555 55555555 55555555 55555555 00003141

$label$ = 41424344

First perform the encryption algorithm: RSAES with (N_B, e_B)

DB = FB2F85C8 8567F3C8 CE9B799C 7C54642D 0C7B41F6 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000155 55555555 55555555 55555555
55555555 55555500 003141

$seed$ = 0D7341FC 5FD510C9 39C75F06 7A9F71E1 E2F4BF40

E = 00342E90 4E85AD5D 51B42142 893FB988 AB37A19E 2DCBA12F
9E1AC1AD 544CDAF0 D695D49A ADD69CDC FAD703D7 26CC9A27
9A5A1770 AB3495E3 3813CEA1 D760B7A8 2C2F0ED4 2FCE0201
E981FCF7 53BCC332 6EC9C29D 0570800C EFFFAB58 F106C5C4
DF6CBD61 44499D8E 8F543864 E872FA30 2D053AB9 D6B4B385
782CAB50 F5389FD0

$C||ID_B$ =
BBF38678 93AE1F37 C0158451 7DF79102 DED224A4 A9342A2E
54CDD67B 41BE24C5 68A32F64 8D299A31 A85DD310 92C9D095
2FFD04B8 02828223 1D375291 B43D244D D6DFBFC2 38334469
4C700138 BD0C8DE6 B9B45B7C 568A7B3E CC84A42D EB79B362
3B04D5EE 22766BD7 B7AC1DAE 00999859 72030664 74B59453
974F8DA9 3C68F8FB FFFF0097

Next perform the signature algorithm: RSA-PSS with (N_A, d_A)

$epsilon$ = 160 (salt has length 160 bits)

tau = 8 (trailer for F is 0xBC, an 8-bit string)

HH = B816546B 167243FB 469596D0 4A9AE844 B559EB6A

lnt = 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00001548 4BD7EC4D 2C793D45
AACC0180 BC7893F1 F34E9

$Mask$ =
528F5F3D B6A5663D 78B905EC 9879F82F EEC83DC9 11819D63
5BA93EA1 36266E11 62085983 7C5C448A 9B8298DE C2ADCF7C
04D44D8A A044828F 8D28F18A 6FA4454E FE01609D 15DAEF62
26D89C06 1B300F98 91BC64E8 933E5C1C 0F3229F7 045DA35A
41E70E61 7726EA66 927CF8

F = 528F5F3D B6A5663D 78B905EC 9879F82F EEC83DC9 11819D63
5BA93EA1 36266E11 62085983 7C5C448A 9B8298DE C2ADCF7C

04D44D8A A044828F 8D28F18A 6FA4454E FE01609D 15DAEF62
26D89C06 1B300F98 91BC64E8 933E5D48 8B8F5733 D69A308E
1B4BCE79 7CE16359 8D4811B8 16546B16 7243FB46 9596D04A
9AE844B5 59EB6ABC

X = 0ACE2D3C 3B5904B7 2750BF08 2171CAC6 AC5F24D4 EC33D1B4
0D1D3A8C CFE1FC9E BF913A92 AF6D8D59 BD44287D B359D605
052F02D9 0D84DB9B 17281170 1830ED77 71FF8B4A B6879544
A2D5DFD4 BFFBABAC C1E12260 8593BE62 16CD3495 5B8872B5
53CFD27D EFABBD0E 2F2CF12F FC790B58 D870ED9D 04CA7ECB
B61D8076 4B96CA44

Bibliography

[1] JEE HEA AN, YEVGENIY DODIS and TAL RABIN, *On the Security of Joint Signature and Encryption*, Advances in Cryptology – EUROCRYPT'02, Lecture Notes in Computer Science, Vol. 2332, pp. 83-107, Springer-Verlag, 2002.

[2] ROBERTO M. AVANZI, *The Complexity of Certain Multi-Exponentiation Techniques in Cryptography*, Journal of Cryptology, Vol. 18, No. 4, pp. 357-373, 2005.

[3] JOONSANG BAEK, RON STEINFELD and YULIANG ZHENG, *Formal Proofs for the Security of Signcryption*, Journal of Cryptology, Vol. 20, No. 2, pp. 203-235, 2007.

[4] YEVGENIY DODIS, MICHAEL J. FREEDMAN, STANISLAW JARECKI and SHABSI WALFISH, *Versatile Padding Schemes for Joint Signature and Encryption*, Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 344-353, 2004.

[5] FIPS 186-3, *Digital Signature Standard*, Federal Information Processing Standard Publication 186-3, US Department of Commerce, National Institute of Standards Technology (NIST), 2009. (Available at http://www.itl.nist.gov/fipspubs/)

[6] ISO/IEC 8825-1:2008, *Information technology — ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.

[7] ISO/IEC 9797-1:2011, *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*.

[8] ISO/IEC 9798-1:2010, *Information technology — Security techniques — Entity authentication — Part 1: General*.

[9] ISO/IEC 10116:2006, *Information technology — Security techniques — Modes of operation for an n-bit block cipher*.

[10] ISO/IEC 10118 (parts 1:2000, 2:2010 & 3:2004), *Information technology — Security techniques — Hash-functions — Part 1: General; Part 2: Hash-functions using an n-bit block cipher; Part 3: Dedicated hash-functions (including Amendment 1:2006)*.

[11] ISO/IEC 11770 (parts 1:2010 & 3:2008), *Information technology — Security techniques — Key management — Part 1: Framework*; *Part 3: Mechanisms using asymmetric techniques.*

[12] ISO/IEC 15946 (parts 1:2008 & 5:2009), *Information technology — Security techniques — Cryptographic techniques based on elliptic curves — Part 1: General; Part 5: Elliptic curve generation*.

[13] ISO/IEC 18032:2005, *Information technology — Security techniques — Prime number generation*.

[14] ISO/IEC 19772:2009, *Information technology — Security techniques — Authenticated encryption*.

[15] DONALD KNUTH, *Seminumerical Algorithms*, Vol. 2 of The Art of Computer Programming, 3rd edition, Addison-Wesley, 1998.

[16] ARJEN LENSTRA, *Key Lengths*, Handbook of Information Security, Hossein Bidgoli (ed.), Vol. 2, pp. 617-635, Wiley, 2005.

[17] JOHN MALONE-LEE and WENBO MAO, *Two Birds One Stone*: *Signcryption Using RSA*, CT-RSA 2003, Lecture Notes in Computer Science, Vol. 2612, pp. 211-225, Springer-Verlag, 2003.

[18] ALFRED MENEZES, TATSUAKI OKAMOTO and SCOTT A. VANSTONE, *Reducing Elliptic Curve Logarithms in a Finite Field*, IEEE Transactions on Information Theory, Vol. IT-39, No. 5, pp. 1639-1646, 1993.

[19] J.-J. QUISQUATER and C. COUVREUR, *Fast decipherment algorithm for RSA public-key cryptosystem*, IEE Electronics Letters, Vol. 18, No. 21, pp. 905-907, 1982.

[20] PETER SHOR, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124-134, IEEE Computer Society Press,1994.

[21] JEROME A. SOLINAS, *Low-Weight Binary Representations for Pairs of Integers*, Technical Report CORR 2001-41, Centre for Applied Cryptographic Research, University of Waterloo, 2001.

[22] WU-CHUAN YANG, D.J. GUAN and CHI SUNG LAIH, *Fast Multicomputation with Asynchronous Strategy*, IEEE Transactions on Computers, Vol. 56, No. 2, pp. 234-242, 2007.

[23] YULIANG ZHENG, *Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption)*, Advances in Cryptology --- Crypto'97, Lecture Notes in Computer Science, Vol. 1294, pp. 165-179, Springer-Verlag, 1997.

[24] YULIANG ZHENG and HIDEKI IMAI, *How to Construct Efficient Signcryption Schemes on Elliptic Curves,* Information Processing Letters, Vol. 68, No. 5, pp. 227-233, 1998.

[25] ISO/IEC 9594 (all parts), *Information technology — Open Systems Interconnection — The Directory*

ICS 35.040

Price based on 53 pages