INTERNATIONAL STANDARD

ISO/IEC 14888-3

Third edition
2016-03-15

Corrected version
2017-09

Information technology — Security techniques — Digital signatures with appendix —

Part 3:
Discrete logarithm based mechanisms

Technologies de l'information — Techniques de sécurité — Signatures numériques avec appendice —

Partie 3: Mécanismes basés sur un logarithme discret

Reference number
ISO/IEC 14888-3:2016(E)

Contents

Page

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

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

This third edition cancels and replaces the second edition (ISO/IEC 14888-3:2006), which has been technically revised. It also incorporates the Amendments ISO/IEC 14888-3:2006/Amd 1:2010 and ISO/IEC 14888-3:2006/Amd 2:2012 and the Technical Corrigenda ISO/IEC 14888-3:2006/Cor 1:2007 and ISO/IEC 14888-3:2006/Cor 2:2009.

This corrected version of ISO/IEC 14888-3:2016 incorporates the following corrections:

— the formula has been changed in 5.1.1.2;

— "G^{x-1}" has been changed to "$G^{x^{-1}}$" in 6.3.1 and 6.3.3;

— "β" has been changed to "β'" in 6.7.1, 6.7.4.4 and 6.7.4.5;

— the reference has been changed in 6.9.1;

— the code for K has been changed in F.9.2.4.

A list of all parts in the ISO/IEC 14888 series can be found on the ISO website.

Introduction

Digital signature mechanisms can be used to provide services such as entity authentication, data origin authentication, non-repudiation and data integrity. A digital signature mechanism satisfies the following requirements.

— Given either or both of the following two things:

 — the verification key, but not the signature key;

 — a set of signatures on a sequence of messages that an attacker has adaptively chosen;

 it should be computationally infeasible for the attacker

 — to produce a valid signature on a new message,

 — in some circumstances, to produce a new signature on a previously signed message, or

 — to recover the signature key;

— it should be computationally infeasible, even for the signer, to find two different messages with the same signature.

NOTE 1 Computational feasibility depends on the specific security requirements and environment.

NOTE 2 In some applications, producing a new signature on a previously signed message without knowing the signature key is allowed. One example of such applications is a membership credential in an anonymous digital signature mechanism as specified in ISO/IEC 20008.

Digital signature mechanisms are based on asymmetric cryptographic techniques and involve the following three basic operations:

— a process for generating pairs of keys, where each pair consists of a private signature key and the corresponding public verification key;

— a process that uses the signature key, called the signature process;

— a process that uses the verification key, called the verification process.

The following are the two types of digital signature mechanisms:

— when, for a given signature key, any two signatures produced for the same message are always identical, the mechanism is said to be deterministic (or non-randomized) (see ISO/IEC 14888-1 for further details);

— when, for a given message and signature key, any two applications of the signature process produce (with high probability) two distinct signatures, the mechanism is said to be randomized (or non-deterministic).

The mechanisms specified in this part of ISO/IEC 14888 are all randomized.

Digital signature mechanisms can also be divided into the following two categories:

— when the whole message has to be stored and/or transmitted along with the signature, the mechanism is termed a "signature mechanism with appendix" (such mechanisms are the subject of ISO/IEC 14888);

— when the whole message, or part of it, can be recovered from the signature, the mechanism is termed a "signature mechanism giving message recovery" (ISO/IEC 9796 specifies mechanisms in this category).

The verification of a digital signature requires access to the signing entity's verification key. It is, thus, essential for a verifier to be able to associate the correct verification key with the signing entity, or more precisely, with (parts of) the signing entity's identification data. This association between the signer's identification data and the signer's public verification key can either be guaranteed by an outside entity or mechanism, or the association can be somehow inherent in the verification key itself. In the former case, the scheme is said to be "certificate-based." In the latter case, the scheme is said to be "identity based." Typically, in an identity-based scheme, the verifier can calculate the signer's public verification key from the signer's identification data. The digital signature mechanisms specified in this part of ISO/IEC 14888 are classified into certificate-based and identity-based mechanisms.

NOTE 3 For certificate-based mechanisms, various PKI standards can be used as the basis of key management. For further information, see ISO/IEC 9594-8 (also known as X.509), ISO/IEC 11770-3 and ISO/IEC 15945.

The security of a signature mechanism is based on an intractable computational problem, i.e. a problem for which, given current knowledge, finding a solution is computationally infeasible, such as the factorization problem and the discrete logarithm problem. This part of ISO/IEC 14888 specifies digital signature mechanisms with appendix based on the discrete logarithm problem, and ISO/IEC 14888-2 specifies digital signature mechanisms with appendix based on the factorization problem.

NOTE 4 The first edition of ISO/IEC 14888 grouped identity-based mechanisms into ISO/IEC 14888-2 and certificate-based mechanisms into ISO/IEC 14888-3, with both parts covering mechanisms based on both the discrete logarithm and the factorization problems. Since the second edition was published, the mechanisms have been reorganized. ISO/IEC 14888-2 now contains integer factoring-based mechanisms, and this part of ISO/IEC 14888 now contains discrete logarithm based mechanisms.

This part of ISO/IEC 14888 includes 12 mechanisms, two of which were in ISO/IEC 14888-3:1998, three of which were from ISO/IEC 15946-2:2002 and three of which were added in ISO/IEC 14888-3:2006. The Elliptic Curve Russian Digital Signature Algorithm (EC-RDSA) and three mechanisms based on Schnorr digital signature are added in ISO/IEC 14888-3:2006/Amd.1:2010.

The mechanisms specified in this part of ISO/IEC 14888 use a collision resistant hash-function to hash the message being signed (possibly in more than one part). ISO/IEC 10118 specifies hash-functions.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14888 may involve the use of patents.

The ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holder of these patent rights has assured the ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information regarding relevant patents is given in the following:

Certicom Corp.

4701 Tahoe Blvd., Building A, Mississauga, ON L4W0B5 Canada.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14888 may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

NOTE 5 The mechanisms of EC-DSA, EC-GDSA. EC-RDSA and EC-FSDSA may be vulnerable to a key substitution attack.[10] The attack is realized if an adversary can find two distinct public keys and one signature such that the signature is valid for both public keys. There are several approaches of avoiding this attack and its possible impact on the security of a cryptographic system. For example, the public key corresponding to the private signing key can be added into the message to be signed.

Information technology — Security techniques — Digital signatures with appendix —

Part 3: Discrete logarithm based mechanisms

1 Scope

This part of ISO/IEC 14888 specifies digital signature mechanisms with appendix whose security is based on the discrete logarithm problem.

This part of ISO/IEC 14888 provides

— a general description of a digital signature with appendix mechanism, and

— a variety of mechanisms that provide digital signatures with appendix.

For each mechanism, this part of ISO/IEC 14888 specifies

— the process of generating a pair of keys,

— the process of producing signatures, and

— the process of verifying signatures.

2 Normative references

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

ISO/IEC 10118-3, *Information technology — Security techniques — Hash-functions*

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

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14888-1 and the following apply.

3.1
finite commutative group
finite set E with the binary operation "*" such that

— for all group elements $a, b \in E$, $a * b \in E$;

— for all group elements $a, b, c \in E$, $(a * b) * c = a * (b * c)$;

— there exists a group element $e \in E$ with $e * a = a$ for all $a \in E$, where e is called the identity element of the group;

— for all group elements $a \in E$, there exists a group element $b \in E$ with $b * a = e$;

— for all group elements $a, b \in E$, $a * b = b * a$

Note 1 to entry: In some cases, such as when E is the set of points on an elliptic curve, arithmetic in the finite set E is described with additive notation.

3.2
cyclic group
finite commutative group (3.1), E, of n elements that contains a group element $a \in E$, called the generator, of order n

3.3
elliptic curve group
cyclic group (3.2) defined on the points of an elliptic curve over a finite field

Note 1 to entry: Let $\boldsymbol{F} = GF(r)$ denote the Galois field with cardinality, r, where either r is an odd prime, p, or r is equal to 2^m, for some positive integer, m. An elliptic curve defined over $\boldsymbol{F}$ can be determined by an affine curve formula, either of the form $y^2 = x^3 + a_1x + a_2$ (when $r = p$ for some odd prime p) or of the form $y^2 + xy = x^3 + a_1x^2 + a_2$ (when $r = 2^m$ for some positive integer m), where the coefficients a_1 and a_2 are (appropriately chosen) elements of $\boldsymbol{F}$. The corresponding elliptic curve E consists of a collection of certain affine points from $\boldsymbol{F} \times \boldsymbol{F}$ together with a special (non-affine) point "at infinity". An affine point P of E is one that can be represented as an ordered pair $(P_x, P_y) \in \boldsymbol{F} \times \boldsymbol{F}$, such that the selection of $x = P_x$ and $y = P_y$ satisfies the given affine curve formula when the indicated arithmetic is performed in the field, $\boldsymbol{F}$. Let "**+**" denote the binary operation known as "elliptic-curve addition", defined for (most) affine points of E by the well-known secant-and-tangent rules. Once the collection of affine points of E is augmented by 0_E, a special point of E "at infinity" that serves as the identity element for "**+**" (but is not represented as an ordered pair), the set E together with the binary operation "**+**" forms a finite, commutative, elliptic-curve group, E.

Note 2 to entry: The cardinality of the elliptic-curve group, E, is one more than the number of ordered pairs in $\boldsymbol{F} \times \boldsymbol{F}$ that satisfy the affine curve formula for E.

3.4
order (of a group element *a*)
least positive integer n such that $a^n=e$, where e is the identity element of the group, a^n is defined recursively such that $a^0=e$ and $a^n=a*a^{m-1}$ $(m>0)$, and * is the group operation

3.5
pairing
function which takes two elements, P and Q, from an *elliptic curve group* (3.3) over a finite field, G_1, as input, and produces an element from another *cyclic group* (3.2) over a finite field, G_2, as output, and which has the following two properties (where it is assumed that the cyclic groups, G_1 and G_2 have order q, for some prime q, and for any two elements P, Q, the output of the pairing function is written as $<P, Q>$)

— Bilinearity: If P, P_1, P_2, Q, Q_1, Q_2 are elements of G_1, and a is an integer satisfying $1 \le a \le q - 1$, then $<P_1 + P_2, Q> = <P_1, Q> * <P_2, Q>$, $<P, Q_1 + Q_2> = <P, Q_1> * <P, Q_2>$, and $<[a]P, Q> = <P, [a]Q> = <P, Q>^a$

— Non-degeneracy: If P is a non-identity element of G_1, $<P, P> \neq 1$

3.6
trusted key generation centre
KGC
trusted third party, which, in an identity-based signature mechanism, generates a private signature key for each signing entity

4 Symbols and abbreviated terms

$a \oplus b$	bitwise exclusive OR of a and b, where a and b are either bits or strings of bits of the same length, and in the latter case, the XOR operation is performed bit-wise
a_1, a_2	elliptic curve coefficients
a mod n	for an arbitrary integer a and a positive integer n, the unique integer remainder r, $0 \le r \le (n - 1)$, satisfying $r = a - bn$, for some integer b.
(A, B, C)	the coefficients of the signature formula, which, for the mechanisms specified in Clause 6, defines how the signature is computed NOTE 1 The signature formula is specified in 5.2.1.
	a parameter which specifies the relationship between the signature key and the verification key
E	an elliptic curve defined by two elliptic curve coefficients, a_1 and a_2
$\mathbb{E}$	a finite commutative group; for the mechanisms based on a multiplicative group, the elements of $\mathbb{E}$ are in $\boldsymbol{Z}_p^*$; for the mechanisms based on an additive group of elliptic curve points, the elements of $\mathbb{E}$ are the points on an elliptic curve E over $GF(r)$
$\#\mathbb{E}$	the cardinality of $\mathbb{E}$; for the mechanisms based on a multiplicative group $\boldsymbol{Z}_p^*$, $\#\mathbb{E}$ is p - 1; for the mechanisms based on an additive group of elliptic curve points, $\#\mathbb{E}$ is one more than the number of points on the elliptic curve E over $GF(r)$ [including 0_E (the point at infinity)]
$\boldsymbol{F}$	a finite field
$\boldsymbol{F}_p$	a finite field of order p
$gcd(N_1, N_2)$	the greatest common divisor of integers N_1 and N_2
G	an element of order q in $\mathbb{E}$
$GF(r)$	the finite field of cardinality r, where r is a prime power
G_1	a cyclic group of prime order q; elements of G_1 are points on an elliptic curve over $GF(r)$
G_2	a cyclic group of prime order q; elements of G_2 are elements of a finite field $GF(r)$
H_1	a hash-function that converts a data string into an element in G_1 NOTE 2 The input data string is converted to an integer first, then the integer is converted to a point on E over $GF(r)$ by using the $I2P$ function, specified in Annex C.
h, H_2	hash-functions, i.e. one of the mechanisms specified in ISO/IEC 10118
ID	a data string containing an identifier of the signer, used in Mechanisms IBS-1 and IBS-2
m	an embedding degree (or extension degree)

$[n]P$	multiplication operation that takes a positive integer n and a point P on the curve E as input and produces as output another point Q on the curve E, where $Q = [n]P = P + P + \ldots + P$ added n -1 times. The operation satisfies $[0]P = 0_E$ (the point at infinity), and $[-n]P = [n](-P)$
P	a generator of G_1 which is used in Mechanisms IBS-1 and IBS-2
p	a prime number or a power of a prime number
q	a prime number that is a divisor of $\#E$ and the order of G_1 and G_2
r	the size of $GF(r)$; in the mechanisms based on an additive group of elliptic curve points, r is a prime power, p^m, for some prime $p \geq 2$ and integer $m \geq 1$.
T	the assignment
T_1	the first part of the assignment T
T_2	the second part of the assignment T
U	the KGC's master private key, generated as a randomly chosen integer, which is used in mechanisms IBS-1 and IBS-2
V	the KGC's master public key, an element G_1, of which is used in mechanisms IBS-1 and IBS-2
$\mathbf{Z}_N^*$	the set of integers i with $0 < i < N$ and $gcd\ (i, N) = 1$, with arithmetic defined modulo N
$\mathbf{Z}_p^*$	the set of integers i with $0 < i < p$ and p a prime number, which is a multiplicative group
α	the bit-length of the prime number (or prime power) p
β	the bit-length of the prime number q
γ	the output bit-length of hash-functions h and H_2
Π	pre-signature
Π_X	x-coordinate of Π in which $\Pi = (\Pi_X, \Pi_Y)$ is an elliptic curve point
Π_Y	y-coordinate of Π in which $\Pi = (\Pi_X, \Pi_Y)$ is an elliptic curve point
Π_a	first element of Π in which $\Pi = (\Pi_a, \Pi_b)$ is an element of an extension field of degree 2
Π_b	second element of Π in which $\Pi = (\Pi_a, \Pi_b)$ is an element of an extension field of degree 2
0_E	the point at infinity on the elliptic curve E
$< >$	a bilinear and non-degenerate pairing
$\|\|$	$X \|\| Y$ is used to mean the result of the concatenation of data items X and Y in the order specified.

5 General model

5.1 Parameter generation process

5.1.1 Certificate-based mechanisms

5.1.1.1 Generation of domain parameters

For digital signature mechanisms based on discrete logarithms, the set of domain parameters includes the following parameters:

— $\mathbb{E}$, a finite commutative group;

— q, a prime divisor of $\#\mathbb{E}$;

— G, an element of order q in $\mathbb{E}$.

In the group $\mathbb{E}$, multiplicative notation is used. It is worthwhile to note that the particular signature mechanism chosen may place additional constraints on the choice of $\mathbb{E}$, q, and G.

5.1.1.2 Generation of signature key and verification key

A signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The corresponding public verification key Y is an element of $\mathbb{E}$ and is computed as

$$Y = G^{X^D}$$

where D is a parameter defined by the mechanism to be used. The value of D is one of two values, -1 and 1.

NOTE An implementation is still considered compliant if it excludes a few integers from consideration as possible X values. For example, the value 1 can be excluded because this value results in the user's verification key being the generator, G, which is easily detectable.

5.1.2 Identity-based mechanisms

5.1.2.1 Notation

The two identity-based mechanisms specified in Clause 7 are both based on the use of pairings over elliptic curve groups. To specify identity-based mechanisms, the additive group notation is used.

5.1.2.2 Generation of domain parameters

The set of domain parameters includes the following parameters:

— $\mathbb{E}$, a finite commutative group;

— $GF(r)$, the Galois field of cardinality r;

— G_1, a cyclic group of prime order q;

— G_2, a cyclic group of prime order q;

— P, a generator of G_1;

— q, a prime number — the cardinality of G_1 and G_2;

— $< >$, a bilinear and non-degenerate pairing

5.1.2.3 Generation of master key

A master private key of a KGC is a secretly generated random or pseudo-random integer U such that $0 < U < q$. The corresponding master public key V is an element of G_1 and is computed as

$$V = [U]P.$$

5.1.2.4 Generation of signature key and verification key

A signature key of a signing entity is an element of G_1 and is computed by the KGC as

$$X = [U]Y$$

where U is the KGC's master private key and $Y = H_1(ID)$ is the public verification key, where ID is an identity string for the KGC and H_1 is a hash-function.

5.1.3 Parameter selection

5.1.3.1 Selecting parameter size

The bit-lengths of parameters for typical security levels are shown in Table 1. The minimum recommended security level is 2^{112}.

NOTE 1 Security level means the number of steps in the best known attack on a cryptographic primitive. If 2^{112} steps are required in the best known attack on a hash-function, the security level of the hash-function is 2^{112}. For a comprehensive analysis of parameter sizes, see References [25] and [34].

It is not necessary to select α, β and γ having the same security level; the security level of an implemented signature scheme is the minimum of the security levels of the parameters.

Table 1 — Parameter sizes according to the security level

Security level	2^{80}	2^{112}	2^{128}	2^{192}	2^{256}
α	1024	2048	3072	7680	15360
β	160	224	256	384	512
γ	160	224	256	384	512

It is recommended that the security level of 2^{80} should only be used for legacy applications.

NOTE 2 Not every mechanism specified in this part of ISO/IEC 14888 provides all of the levels of security specified in this table. For example, DSA in 6.1 only supports the security levels up to 2^{128}.

5.1.3.2 Selecting a hash-function

Selection of hash-functions should be based on those standardized in ISO/IEC 10118-3. That is, h and H_2 shall be one of the mechanisms specified in ISO/IEC 10118-3, and H_1 converts a data string obtained by using one of the mechanisms specified in ISO/IEC 10118-3 into an element in G_1.

The hash-functions used in this part of ISO/IEC 14888 should be collision-resistant.

The security strength for the selected hash-function should meet or exceed the security strength of the parameters used in key generation. The relationship between the security levels of a hash-function and the key generation parameters is shown in 5.1.3.1.

Furthermore, implementations that verify digital signatures shall have a way of securely determining which hash-function was used by the signer. Otherwise, an attacker might be able to convince a verifier to use a different weaker, hash-function and thus, bypass the intended security level.

5.1.4 Validity of domain parameters and verification key

The signature verifier may require assurance that the domain parameters and public verification key are valid, otherwise, there is no assurance of meeting the intended security even if the signature verifies, and an adversary may be able to generate signatures that verify.

Assurance of the validity of domain parameters can be provided by one of the following:

— selection of valid domain parameters from a trusted published source, such as a standard;

— generation of valid domain parameters by a trusted third party, such as a CA or a KGC;

— validation of candidate domain parameters by a trusted third party, such as a CA or a KGC;

— for the signer, generation of valid domain parameters by the signer using a trusted system;

— validation of candidate domain parameters by the user (i.e. the signer or verifier).

Assurance of validity of a public verification key can be provided by one of the following:

— for the signer, generation of the public verification/private signature key pair using a trusted system;

— for the signer or verifier, validation of the public verification key by a trusted third party, such as a CA or a KGC;

— validation of the public verification key by the user (i.e. the signer or verifier).

NOTE 1 Validation of domain parameters and keys is required. However, how to achieve this is outside the scope of this part of ISO/IEC 14888.

NOTE 2 The method of authenticating the signer is dependent on the real applications, which is out of the scope of this part of ISO/IEC 14888.

5.2 Signature process

5.2.1 General

All of the signature mechanisms in this part of ISO/IEC 14888 make use of a randomizing value K, which is used (along with the message) to produce a witness R (the first part of the signature) and an assignment (T_1, T_2). The signature for the message is the pair (R, S) where S (the second part of the signature) is computed as the solution of a signature formula.

In the certificate-based mechanisms, specified in Clause 6, the signature formula is

$$AK + BX^D + C \equiv 0 \pmod{q},$$

given that (A, B, C) is a permutation of (S, T_1, T_2), X is the private signature key and D is a parameter depending on the particular mechanism.

In the identity-based mechanisms specified in Clause 7, the signature formula is

$$[K]A + [U^D]B + C \equiv 0_E \text{ (in } G_1\text{)}.$$

Given that (A, B, C) is a permutation of (S, T_1, T_2), U is the master private key and D is a parameter depending on the particular mechanism.

The permutation will be specified or agreed upon when setting up the signature system.

The signature process and the formation of a signed message consist of the following eight stages (see Figure 1):

— producing the randomizer;

— producing the pre-signature;

— preparing the message for signing;

— computing the witness;

— computing the assignment (it is not necessary to compute the assignment in the identity-based mechanisms);

— computing the second part of the signature;

— constructing the appendix;

— constructing the signed message.

In this process, the signing entity makes use of its private signature key, its public verification key (optional) and the domain parameters.

5.2.2 Producing the randomizer

For each signature, the signing entity freshly generates a secret randomizer which is an integer K with $0 < K < q$. The output of this stage is K, which shall be kept secret and destroyed safely after use.

NOTE 1 The randomizer K can be considered as an ephemeral key.

NOTE 2 For the same rationale of 5.1.1.2, an implementation is still considered compliant if it excludes a few integers from consideration as possible K values.

5.2.3 Producing the pre-signature

The inputs to this stage are the randomizer K and optionally signature key X, with which the signing entity computes the pre-signature, Π , by using K and public parameters as input. In the certificate-based mechanisms specified in Clause 6, it is computed as

$$\Pi = G^K$$

in $\mathbb{E}$. In the identity-based mechanisms specified in Clause 7, it is individually specified in the mechanisms. The output of this stage is the pre-signature, Π .

5.2.4 Preparing the message for signing

In the process of preparing the message, one of M_1 and M_2 becomes message M, the other becomes empty.

5.2.5 Computing the witness (the first part of the signature)

The variables to this stage are the pre-signature Π from 5.2.3 and M_1 from 5.2.4. The values of these variables are taken as inputs to the witness function. The output of the witness function is the witness R. The witness function is specified in the mechanisms.

5.2.6 Computing the assignment

The inputs to the assignment function are the first part of the signature, which is the witness R from 5.2.5, M_2 from 5.2.4, and, optionally, the verification key Y. The output of the assignment function is assignment $T = (T_1, T_2)$. In the certificate-based mechanisms specified in Clause 6, T_1 and T_2 are integers such that

$$0 < |T_1| < q, 0 < |T_2| < q.$$

In the identity-based mechanisms specified in Clause 7, T_1 and T_2 are elements of G_1. It is not necessary to compute T in the identity-based mechanisms.

5.2.7 Computing the second part of the signature

The inputs to this stage are the randomizer K from 5.2.1, the signature key X, the assignment $T = (T_1, T_2)$ from 5.2.6, the permutation (A, B, C) of (S, T_1, T_2), a variable D in 5.1.1.2 and domain parameter q as specified in 5.1.1.1 and 5.1.2.1.

In the certificate-based mechanisms, the signing entity forms the signature formula

$$AK + BX^D + C \equiv 0 \pmod q$$

and solves the signature formula for S, the second part of the signature, where $0 < S < q$.

In the identity-based mechanisms, the signing entity solves the signature formula for S, the second part of the signature such that $S \in G_1$. This solution satisfies the signature formula

$$[K]A + [U^D]B + C \equiv 0_E \text{ (in } G_1).$$

The pair (R, S) will be called the signature, Σ.

5.2.8 Constructing the appendix

The appendix is constructed from the signature and an optional text field, *text*, as [(R, S), *text*]. The text field could include a certificate that cryptographically ties the public verification key to the identification data of the signing entity.

As indicated in ISO/IEC 14888-1, depending on the application, there are different ways of forming the appendix and appending it to the message. The general requirement is that the verifier is able to relate the correct signature to the message. For successful verification, it is also essential that prior to the verification process, the verifier is able to associate the correct verification key with the signature.

5.2.9 Constructing the signed message

The signed message is obtained by the concatenation of message M and the appendix, i.e. M || [(R, S), *text*].

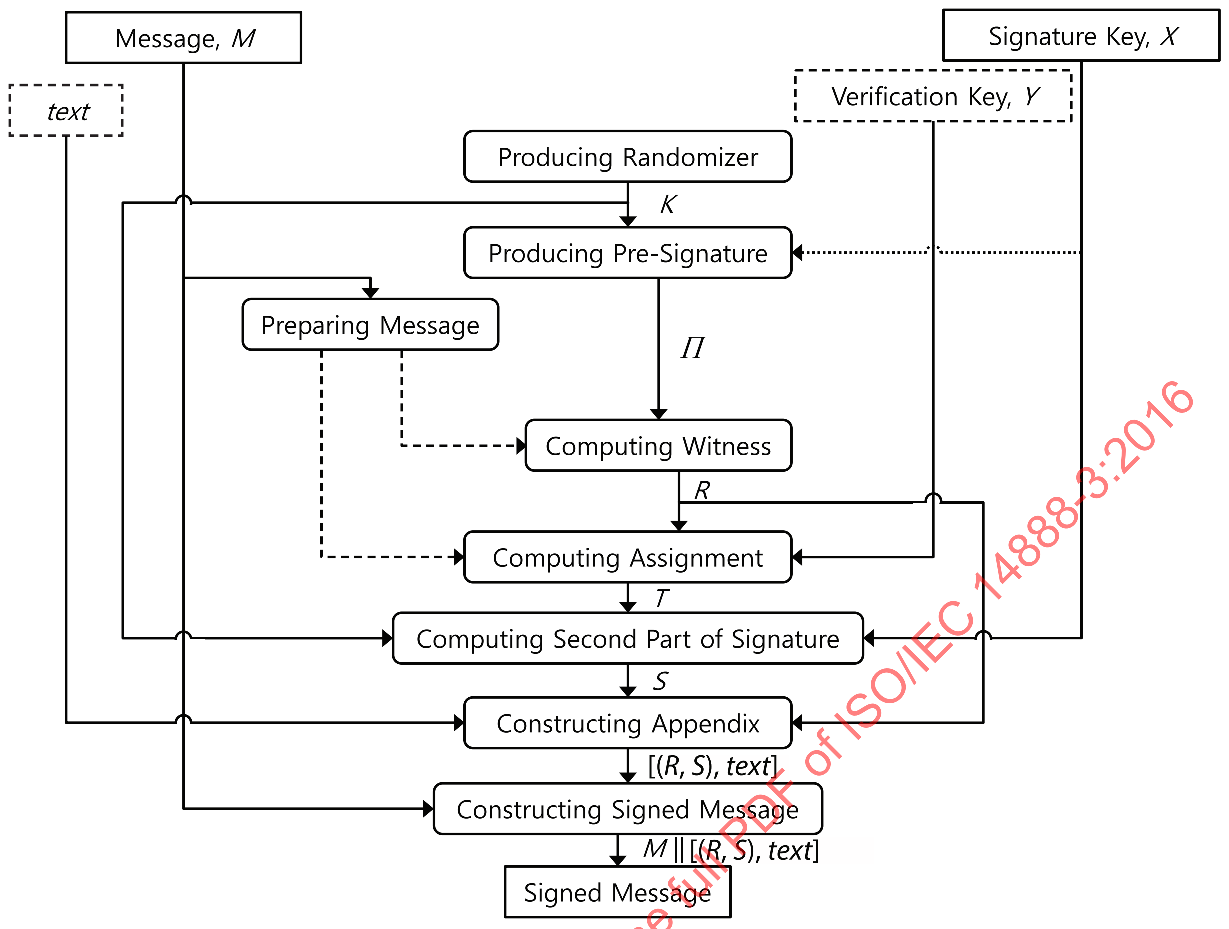

Figure 1 — Signature process with randomized witness (one of M_1 and M_2 is M, the other is empty)

5.3 Verification process

5.3.1 General

The verification process consists of the following six stages (see Figure 2):

— retrieving the witness;

— preparing message for verification;

— retrieving the assignment (it is optional to compute the assignment in the identity-based mechanisms);

— recomputing the pre-signature;

— recomputing the witness;

— verifying the witness.

In this process, the verifier makes use of the signer's verification key, KGC's master public key (only for the identity-based mechanisms specified in Clause 7) and the domain parameters.

5.3.2 Retrieving the witness

The verifier retrieves the signature (R, S) from the appendix, and divides it into the witness R and the second part of the signature S. Also, the verifier checks the range or the bit length of the signature elements, R and S, according to the rule specified by each signature process. If the predefined rule is violated, the signature shall be rejected.

5.3.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2.

5.3.4 Retrieving the assignment

This stage is identical to 5.2.6. The inputs to the assignment function consist of the witness R from 5.3.2, M_2 from 5.3.3, and (optionally) the verification key Y. The assignment $T = (T_1, T_2)$ is recomputed as the output from the assignment function. In the identity-based mechanisms, it is not necessary to recompute T.

5.3.5 Recomputing the pre-signature

The inputs to this stage are the set of domain parameters, the verification key Y, the assignment $T = (T_1, T_2)$ from 5.3.4, the second part of the signature S from 5.3.2, and optionally R from 5.3.2. The verifier assigns to the coefficients (A, B, C) the values (S, T_1, T_2) according to the order specified by the signature function, and in the certificate-based mechanisms, computes the element Π'.

In the certificate-based mechanisms, Π' is computed in E as

$$\Pi' = Y^m G^n$$

where $m = -A^{-1}B \bmod q$ and $n = -A^{-1}C \bmod q$.

In the identity-based mechanisms, it is individually specified in the mechanisms.

5.3.6 Recomputing the witness

The computations at this stage are the same as in 5.2.5. The verifier executes the witness function. The inputs are Π' from 5.3.5 and M_1 from 5.3.3. The output is the recomputed witness, R'.

In Mechanism IBS-2 specified in 7.2, the process of recomputing the witness is computing two verification functions, instead of computing R'.

5.3.7 Verifying the witness

The signature is verified if the recomputed witness, R', from 5.3.6 is equal to R from 5.3.2.

In Mechanism IBS-2 specified in 7.2, the process of verifying the witness involves checking whether the two verification function values computed in 5.3.6 are identical, instead of verifying whether $R = R'$ holds.

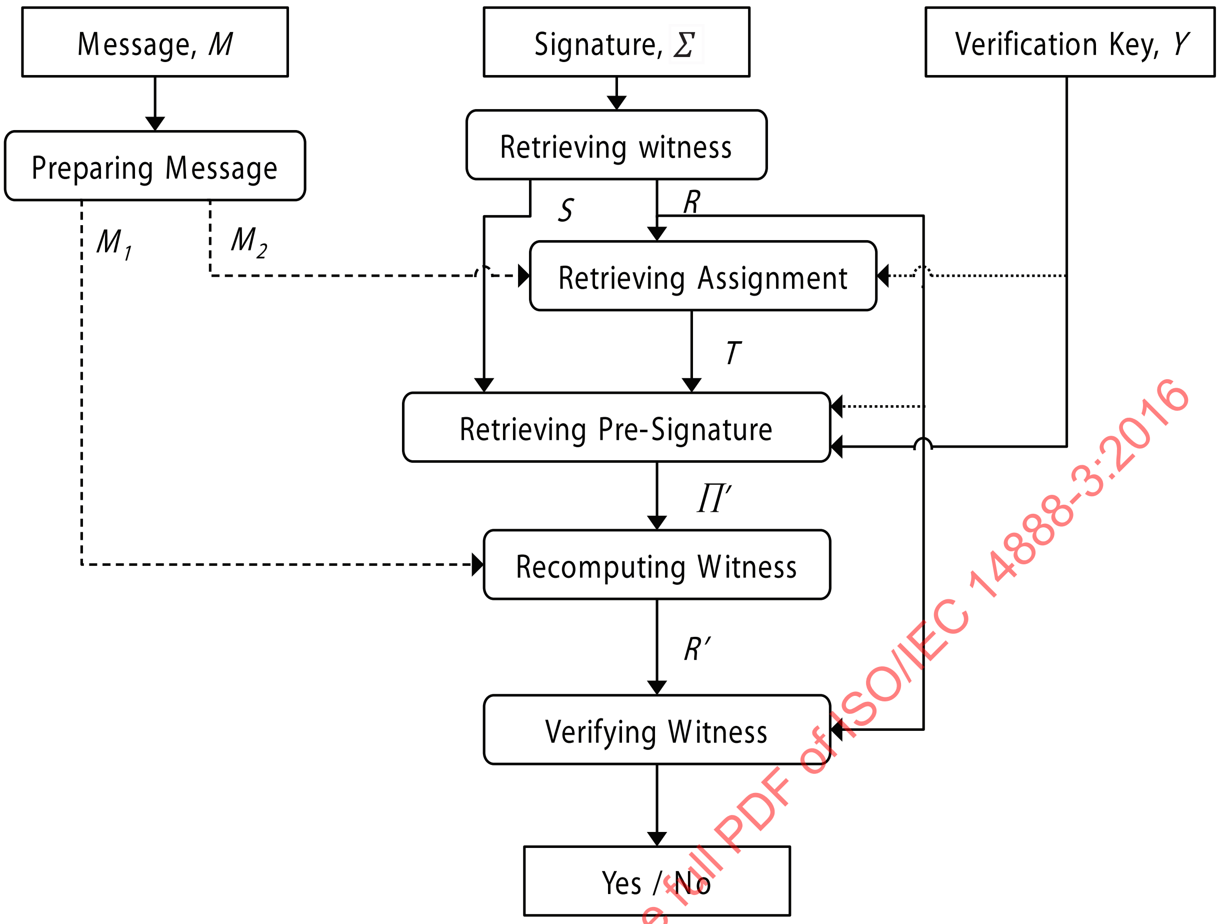

Figure 2 — Verification process with a randomized witness

6 Certificate-based mechanisms

6.1 General

6.1 6.1 General

In elliptic curve arithmetic, an elliptic curve point is represented as affine coordinates. That is, an elliptic curve point Π has two coordinates: x-coordinate, Π_X, and y-coordinate, Π_Y. Elliptic curves for EC-DSA, EC-KCDSA, EC-GDSA, EC-RDSA, EC-SDSA and EC-FSDSA are restricted to non-singular and non-supersingular curves.

The hash-function identifier can be used for binding the signature mechanism and the hash-function.

6.2 DSA

6.2.1 General

DSA (Digital Signature Algorithm) is a signature mechanism with $\boldsymbol{E} = \boldsymbol{Z}_p^*$, p a prime, and q a prime dividing $p - 1$. The parameter D of DSA is equal to 1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness function is defined by the formula

$$R = \Pi \bmod q, \text{ where } \Pi = G^k \bmod p \text{ for some integer } K.$$

and the assignment function by the formula

$$(T_1, T_2) = (-R, -BS2I(\gamma, H)),$$

where $H = h(M)$ is the truncated hash-code of message M, converted to an integer according to the conversion rule given in Annex B.

The coefficients (A, B, C) of the DSA signature formula are set as follows:

$$(A, B, C) = (S, T_1, T_2).$$

Thus the signature formula becomes

$$SK - RX - BS2I(\gamma, H) \equiv 0 \pmod q.$$

NOTE This mechanism is taken from Reference [17]. The notation here has been changed slightly from Reference [17] to conform with the notation used elsewhere in this part of ISO/IEC 14888.

6.2.2 Parameters

p	a prime, where $2^{\alpha-1} < p < 2^{\alpha}$.
q	a prime divisor of p -1, where $2^{\beta-1} < q < 2^{\beta}$.
G	a generator of the subgroup of order q, such that $1 < G < p$.

Four choices for the pair (α, β) are allowed in DSA, which are (1024, 160), (2048, 224), (2048, 256), and (3072, 256). It is recommended that the security strength of the (α, β) pair and the γ be the same unless an agreement has been made between participating entities to use a stronger hash function.

The integers p, q, and G can be public and can be common to a group of users.

The parameters p, q and G are generated as specified in Annex D. If compatibility with the NIST Federal standard is not required then the parameters p and q can be generated by using the prime generation techniques given in ISO/IEC 18032.

It is recommended that all users check the proper generation of the DSA public parameters according to Reference [17].

NOTE If a signer were free to specifically choose the domain parameter q to facilitate a collision between hash values, then an attack against such an instance of DSA could be mounted in which the required collision on the underlying hash-function could be found with complexity of 2^{74}, 2^{101}, or 2^{114} (corresponding to γ = 160, 224, or 256, respectively), as opposed to the most secure instances, in which the complexity of finding a collision would be 2^{80}, 2^{112}, or 2^{128}.[39] The attack is, however, easily detectable. Furthermore, the attack cannot be mounted when domain parameters are generated as specified in Reference [17] which includes the method specified in Annex D. If the use of an appropriate method for the generation of domain parameters cannot be verified, the attack can also be prevented by using a mechanism of the type specified in 6.3, 6.4 and 6.7.

It is recommended that digital signatures based on SHA-1 should only be used for legacy applications.

6.2.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is

$$Y = G^X \bmod p.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.2.4 Signature process

6.2.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.2.4.2 Producing the pre-signature

The input to this stage is the randomizer K, and the signing entity computes

$$\Pi = G^K \bmod p.$$

6.2.4.3 Preparing the message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.2.4.4 Computing the witness

The signing entity computes $R = \Pi \bmod q$ where the witness is simply a function of the pre-signature. Thus

$$R = (G^K \bmod p) \bmod q.$$

6.2.4.5 Computing the assignment

The signing entity computes the hash-code; if the output bit length of the selected hash-function is larger than $\lceil \log_2 q \rceil$, H is set to the leftmost (most significant) $\lceil \log_2 q \rceil$ bits of $h(M_2)$. Otherwise, H is $h(M_2)$. Afterwards, H is converted to integer according to conversion rule, *BS2I,* in Annex B. The assignment (T_1, T_2) is $(-R, -BS2I(\gamma, H))$.

6.2.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed in 6.2.4.4 and

$$S = (K^{-1}(BS2I(\gamma, H) + XR)) \bmod q$$

where H = the leftmost (most significant) $\min(\beta, \gamma)$ bits of $h(M_2)$.

The value of $h(M_2)$ is an γ-bit string output of the appropriate hash-function in 6.2.2. For use in computing S, this string shall be converted to an integer.

It is required to check if $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated (it is extremely unlikely that $R = 0$ or $S = 0$ if signatures are generated properly).

6.2.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.2.4.8 Constructing the signed message

A signed message is the concatenation of a message, M, and the appendix.

$$M||((R, S), text)$$

6.2.5 Verification process

6.2.5.1 General

Prior to verifying the signature of a signed message, it is necessary that the verifier has trusted copies of p, q, G and Y.

6.2.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. And, the verifier checks to see that $0 < R < q$ and $0 < S < q$. If either condition is violated, the signature shall be rejected.

6.2.5.3 Preparing the message for verification

The verifier retrieves $M_2 = M$ from the signed message. M_1 is empty.

6.2.5.4 Retrieving the assignment

This stage is identical to 6.2.4.5. The inputs to the assignment function consist of the witness R from 6.2.5.2 and M_2 from 6.2.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.2.4.5.

6.2.5.5 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.2.5.4 and second part of the signature S from 6.2.5.2. The verifier obtains a recomputed value Π' of the pre-signature using the formula

$$\Pi' = Y^{-S^{-1}T_1 \bmod q} G^{-S^{-1}T_2 \bmod q} \bmod p.$$

6.2.5.6 Recomputing the witness

The computations at this stage are the same as in 6.2.4.4. The verifier executes the witness function. The input is Π' from 6.2.5.5. Note that M_1 is empty. The output is the recomputed witness R' such that $R' = \Pi' \bmod q$.

6.2.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.2.5.6 to the value of R from 6.2.5.2. If $R' = R$, then the signature is verified.

6.3 KCDSA

6.3.1 General

KCDSA (Korean Certificate-based Digital Signature Algorithm) is a signature mechanism with $E = \mathbf{Z}_p^*$, p a prime, and q a prime dividing $p - 1$. Verification key Y is $G^{x^{-1}}$; that is, the parameter D is -1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness function is defined by the formula

$$R = h(I2BS(\beta, \Pi))$$.

If γ is longer than β, then the witness function is replaced by the formula

$$R = I2BS(\beta, BS2I(\gamma, h(I2BS(\beta, \Pi))) \bmod 2^{\beta})$$.

Domain parameters shall indicate the employed hash-function. The assignment function is defined by the formula

$$(T_1, T_2) = (V, -1),$$

where $V = BS2I(\beta, R \oplus H) \bmod q$. The value H is the hash-code from the public key Y and message M.

The coefficients (A, B, C) of the KCDSA signature formula are set as,

$$(A, B, C) = (T_2, S, T_1).$$

Thus the signature formula becomes

$$-K + SX^{-1} + V \equiv 0 \pmod q).$$

NOTE This mechanism is taken from Reference [36]. The notation here has been changed slightly from Reference [36] to conform with notation used elsewhere in this part of ISO/IEC 14888.

6.3.2 Parameters

p	a prime, where $2^{\alpha-1} < p < 2^{\alpha}$
q	a prime divisor of $p - 1$, where $2^{\beta-1} < q < 2^{\beta}$
F	an integer such that $1 < F < p - 1$ and $F^{(p-1)/q} \bmod p > 1$
G	$F^{(p-1)/q} \bmod p$, an element of order q in $\mathbf{Z}_p^*$
l	the input block size (in bits) of the selected hash-function h

Hash-function identifier or OID with specified hash-function.

Three choices of the triplet (α, β, h) are allowed in KCDSA, which are (2048, 224, SHA-224), (3072, 256, SHA-256), and (2048, 224, SHA-256). Among these, (2048, 224, SHA-224) and (3072, 256, SHA-256) are recommended, while (2048, 224, SHA-256) can be used in the case when only SHA-256 is available and SHA-224 is not.

The integers, p, q, G and l, can be public and can be common to a group of users.

6.3.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is –1. The corresponding public verification key Y is

$$Y = G^{X^{-1}} \bmod p\,.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.3.4 Signature process

6.3.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.3.4.2 Producing the pre-signature

The input to this stage is the randomizer K and the signing entity computes

$$\Pi = G^K \bmod p.$$

6.3.4.3 Preparing the message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.3.4.4 Computing the witness

The signing entity computes witness $R = h(I2BS(\beta, \Pi))$, where the output of H is the hash-code of the bit string of length α converted from the pre-signature Π. If γ is longer than β, then, the computation of witness is replaced by $R = I2BS(\beta, BS2I(\gamma, h(I2BS(\beta, \Pi))) \bmod 2^\beta)$.

The conversion rules, *I2BS* and *BS2I*, are given in Annex B.

6.3.4.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (V, -1)$ where $V = BS2I(\beta, R \oplus H) \bmod q$, where $H{=}h(Y' || M_2)$ is the hash-code of the concatenation of $Y' = I2BS(l, Y \bmod 2^l)$ and message M_2. The value of Y' is a bit string of length l. In computing V, the bit string $R \oplus H$ shall be converted to an integer before modulo reduction with respect to q.

If γ is longer than β, then the computation of H is replaced by $H = I2BS(\beta, BS2I(\gamma, h(Y' || M_2)) \bmod 2^\beta)$.

NOTE Y' is a fixed value for a user, thus, this value can be kept as a user parameter.

6.3.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed in 6.3.4.4 and

$$S = X(K - V) \bmod q.$$

6.3.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.3.4.8 Constructing the signed message

A signed message is the concatenation of a message, M, and the appendix.

$$M||((R, S), text)$$

6.3.5 Verification process

6.3.5.1 General

Prior to verifying the signature of a signed message, it is necessary that the verifier has trusted copies of p, q and G.

The verifier also acquires the necessary data items for the verification process: for example, the verification key Y (see ISO/IEC 14888-1:2008, Clause 9 for additional required data items).

6.3.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then checks whether the following conditions hold or not:

— $0 < S < q$;

— If the length of the value γ is not longer than β, the bit length of R is equal to the output bit length of the employed hash-function h;

— If the length of the value γ is longer than β, the bit length of R is equal to β.

If any of the above condition does not hold the signature shall be rejected.

6.3.5.3 Preparing the message for verification

The verifier retrieves $M_2 = M$ from the signed message. M_1 is empty.

6.3.5.4 Retrieving the assignment

This stage is identical to 6.3.4.5. The inputs to the assignment function consist of the witness R from 6.3.5.2 and M_2 from 6.3.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.3.4.5.

6.3.5.5 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.3.5.4 and second part of the signature S from 6.3.5.2. The verifier obtains a recomputed value Π' of the pre-signature using the formula

$$\Pi' = Y^{S \bmod q} G^{T1 \bmod q} \bmod p.$$

6.3.5.6 Recomputing the witness

The computations at this stage are the same as in 6.3.4.4. The verifier executes the witness function. The input is Π' from 6.3.5.5. Note that M_1 is empty. The output is the recomputed witness R'.

6.3.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.3.5.6 to the value of R from 6.3.5.2. If $R' = R$, then the signature is valid.

6.4 Pointcheval/Vaudenay algorithm

6.4.1 General

The method of Pointcheval/Vaudenay is a variant of the DSA algorithm, with $E = \mathbf{Z}_p^*$, p a prime, and q a prime divisor of $p - 1$. The parameter D is equal to 1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness is defined by the formula

$$R = \Pi \bmod q$$

and the assignment function is defined by the formula

$$(T_1, T_2) = (-R, -H)$$

where $H = h(I2BS(\beta, R)||M)$ is the hash-code of the concatenation of the witness R and the message M. Note that the computation of T_2 above requires the conversion of the hash-code to an integer. The conversion function is given in Annex B.

The coefficients (A, B, C) of the Pointcheval/Vaudenay signature formula are set as follows:

$$(A, B, C) = (S, T_1, T_2).$$

Thus, the signature formula becomes

$$SK - RX - H \equiv 0 \pmod q.$$

NOTE This mechanism is based on the algorithm designed by D. Pointcheval and S. Vaudenay in Reference [31].

6.4.2 Parameters

p	a prime
q	a prime divisor of $p - 1$
F	an integer such that $1 < F < p - 1$ and $F^{(p-1)/q} \bmod p > 1$
G	$F^{(p-1)/q} \bmod p$

Hash-function identifier or OID with specified hash-function

It is recommended that all users check the proper generation of the public parameters.

6.4.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is

$$Y = G^X \bmod p.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.4.4 Signature process

6.4.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.4.4.2 Producing the pre-signature

The input to this stage is the randomizer K and the signing entity computes

$$\Pi = G^K \bmod p.$$

6.4.4.3 Preparing message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.4.4.4 Computing the witness

The signing entity computes $R = \Pi \bmod q$ where the witness is simply a function of the pre-signature. Thus

$$R = (G^K \bmod p) \bmod q.$$

6.4.4.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (-R, -BS2I(\gamma, H))$, where $H = h(I2BS(\beta, R)||M_2)$ is the hash-code of the concatenation of the witness and message M_2. Before the concatenation, the witness shall be converted to a bit string of length $|p|$.

6.4.4.6 Computing the signature

The signature is (R, S) where R is computed in 6.4.4.4 and

$$S = K^{-1}(BS2I(\gamma, H) + XR) \bmod q.$$

6.4.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.4.4.8 Constructing the signed message

A signed message is the concatenation of a message, M, and the appendix.

$$M||((R, S), text)$$

6.4.5 Verification process

6.4.5.1 General

Prior to verifying the signature of a signed message, it is necessary that the verifier has trusted copies of p, q and G.

The verifier also acquires the necessary data items for the verification process: for example, the verification key Y (see ISO/IEC 14888-1:2008, Clause 9 for additional required data items).

6.4.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then checks to see that $0 < R < q$ and $0 < S < q$. If either condition is violated, the signature shall be rejected.

6.4.5.3 Preparing the message for verification

The verifier retrieves $M_2 = M$ from the signed message. M_1 is empty.

6.4.5.4 Retrieving the assignment

This stage is identical to 6.4.4.5. The inputs to the assignment function consist of the witness R from 6.4.5.2 and M_2 from 6.4.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.4.4.5.

6.4.5.5 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.4.5.4 and second part of the signature S from 6.4.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = Y^{-S^{-1}T_1 \bmod q}\, G^{-S^{-1}T_2 \bmod q} \bmod p.$$

6.4.5.6 Recomputing the witness

The computations at this stage are the same as in 6.4.4.4. The verifier executes the witness function. The inputs are Π' from 6.4.5.5. Note that M_1 is empty. The output is the recomputed witness R'.

6.4.5.7 Verifying the witness

The verifier compares the recomputed witness, R', from 6.4.5.6 to the value of R from 6.4.5.2. If $R' = R$, then the signature is valid.

6.5 SDSA

6.5.1 General

SDSA (Schnorr Digital Signature Algorithm) is a signature mechanism with $E = \mathbf{Z}_p^*$, p a prime, and q a prime dividing p-1. The parameter D is equal to 1. The message is prepared such that M_1 is the message to be signed, i.e. $M_1 = M$, and M_2 is empty. The witness function is defined by setting R equal to a hash-code. The assignment function is defined by setting T_1 = -1 and T_2 equal to the negative of the integer which is obtained by converting R to an integer according to the conversion rule, *BS2I*, given in Annex B and then reducing modulo q.

The coefficients (A, B, C) of the SDSA signature formula are set as follows:

$$(A, B, C) = (T_1, T_2, S).$$

Thus the signature formula becomes

$$-K + T_2X + S \equiv 0 \pmod{q}.$$

NOTE SDSA stands for Schnorr Digital Signature Algorithm. The mechanism is taken from References [32] and [33]. The notation here has been changed from References [32] and [33] to conform with the notation used in ISO/IEC 14888.

6.5.2 Parameters

p	a prime, where $2^{\alpha-1} < p < 2^{\alpha}$.
q	a prime divisor of p -1, where $2^{\beta-1} < q < 2^{\beta}$.
G	a generator of the subgroup of order q, such that $1 < G < q$.

Four choices for the pair (α, h) are allowed in SDSA, namely (1024, SHA-1), (2048, SHA-224), (2048, SHA-256), and (3072, SHA-256). Corresponding β should be selected according to α in Table 1.

The integers p, q, and G can be public and can be common to a group of users.

The parameters p, q and G are generated as specified in Annex D. The parameters p and q can be generated using the prime generation techniques given in ISO/IEC 18032.

It is recommended that all users check the proper generation of the SDSA public parameters according to Reference [17].

It is recommended that digital signatures based on SHA-1 should only be used for legacy applications.

6.5.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is

$$Y = G^X \bmod p.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.5.4 Signature process

6.5.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.5.4.2 Producing the pre-signature

The input to this stage is the randomizer K, and the signing entity computes

$$\Pi = G^K \bmod p.$$

6.5.4.3 Preparing message for signing

The message is prepared such that M_1 is the message to be signed, i.e. $M_1 = M$, and M_2 is empty.

6.5.4.4 Computing the witness

The signing entity computes the witness R as the hash-code of the pre-signature Π and the first part of the message M_1

$$R = h(I2BS(\beta, \Pi) \,||M).$$

6.5.4.5 Computing the assignment

The value of the witness R is converted to an integer according to conversion rule, *BS2I,* in Annex B and then reducing modulo q. The assignment ($T1$, $T2$) is (-1, $-BS2I(\gamma, R) \bmod q$).

6.5.4.6 Computing the second part of the signature

The signature is (R, S) where $S = (K + BS2I(\gamma, R)X) \bmod q$.

As an option, one may wish to check if $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated (it is extremely unlikely that $R = 0$ or $S = 0$ if signatures are generated properly).

6.5.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field *text*.

6.5.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix

$M \,||\, ((R,S) \,||\, text)$.

6.5.5 Verification process

6.5.5.1 General

The verifying entity acquires the necessary data items required for the verification process.

6.5.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier checks to see that R is a non-zero string within the range of the hash function and that $0 < S < q$.

6.5.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2, such that $M_1 = M$ and M_2 is empty.

6.5.5.4 Retrieving the assignment

The input to the assignment function consists of the witness R from 6.5.5.2. The assignment

$T = (T_1, T_2) = (-1, -BS2I(\gamma, R) \bmod q)$.

6.5.5.5 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.5.5.4 and second part of the signature S from 6.5.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = Y^{(T2 \bmod q)} G^{(-ST1 \bmod q)} \bmod p = Y^{(T2 \bmod q)} G^{(S \bmod q)} \bmod p.$$

6.5.5.6 Recomputing the witness

The computations at this stage are the same as in 6.5.4.4 and 6.5.4.5. The verifier executes the witness function. The input is Π' from 6.5.5.5 and M_1 from 6.5.5.3. The output is the recomputed witness R' which is the hash-code of the recomputed pre-signature Π' and the message M.

6.5.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.5.5.6 to the retrieved version of R from 6.5.5.2. If $R' = R$, then the signature is verified.

6.6 EC-DSA

6.6.1 General

EC-DSA (Elliptic Curve Digital Signature Algorithm) is an elliptic curve analogue of the DSA algorithm. The coefficients (A, B, C) of the EC-DSA signature formula are set as follows

$$(A, B, C) = (S, T_1, T_2)$$

where $(T_1, T_2) = (-R, -BS2I(\gamma, H))$ and $H = h(M)$ is the truncated hash-code of message M, converted to an integer according to the conversion rule given in Annex B. The hash-function h is one of SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 described in ISO/IEC 10118-3.

Verification key Y is $[X]G$; that is, the parameter D is equal to 1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness function is defined by the formula

$$R = FE2I(r, \Pi_X) \bmod q.$$

The conversion rule, *FE2I*, is given in Annex B.

Thus the signature formula becomes

$$SK - RX - BS2I(\gamma, H) \equiv 0 \pmod{q}.$$

NOTE This mechanism is based on the algorithm described in Reference [7].

6.6.2 Parameters

F	a finite field
E	an elliptic curve group over field $\boldsymbol{F}$
$\#E$	the cardinality of E
q	a prime divisor of $\#E$
G	a point on the elliptic curve of order q

All these parameters can be public and can be common to a group of users. The security strength of the hash function used shall meet or exceed the security strength associated with the bit length of q.

It is recommended that all users check the proper generation of the EC-DSA public parameters according to Reference [7] or Reference [17].

It is recommended that digital signatures based on SHA-1 should only be used for legacy applications

6.6.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is

$$Y = [X]G.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.6.4 Signature process

6.6.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.6.4.2 Producing the pre-signature

The input to this stage is the randomizer K and the signing entity computes

$$\Pi = [K]G.$$

6.6.4.3 Preparing message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.6.4.4 Computing the witness

The signing entity computes $R = FE2I(r, \Pi_X) \bmod q$.

6.6.4.5 Computing the assignment

The signing entity computes the hash-code; if the output bit length of the selected hash-function is larger than $\lceil \log_2 q \rceil$, H is set to the leftmost (most significant) $\lceil \log_2 q \rceil$ bits of $h(M_2)$. Otherwise, H is $h(M_2)$. Afterwards, H is converted to integer according to conversion rule, $BS2I$, in Annex B. The assignment (T_1, T_2) is $(-R, -BS2I(\gamma, H))$.

6.6.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed in 6.6.4.4, and

$$S = (K^{-1}(XR + BS2I(\gamma, H))) \bmod q.$$

It is required to check if $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated (it is extremely unlikely that $R = 0$ or $S = 0$ if signatures are generated properly).

6.6.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.6.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix.

$$M||((R, S), text)$$

6.6.5 Verification process

6.6.5.1 General

The verifying entity acquires the necessary data items required for the verification process.

6.6.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then first checks to see that $0 < R < q$ and $0 < S < q$; if either condition is violated the signature shall be rejected.

6.6.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2. M_1 will be empty and $M_2 = M$.

6.6.5.4 Retrieving the assignment

This stage is identical to 6.6.4.5. The inputs to the assignment function consist of the witness R from 6.6.5.2 and M_2 from 6.6.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.6.4.5.

6.6.5.5 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.6.5.4 and second part of the signature S from 6.6.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = [-S^{-1}T_1 \bmod q]Y + [-S^{-1}T_2 \bmod q]G.$$

6.6.5.6 Recomputing the witness

The computations at this stage are the same as in 6.6.4.4. The verifier executes the witness function. The input is Π' from 6.6.5.5. The output is the recomputed witnessR'.

6.6.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.6.5.6 to the retrieved version of R from 6.6.5.2. If $R' = R$, then the signature is verified.

6.7 EC-KCDSA

6.7.1 General

EC-KCDSA (Elliptic Curve Korean Certificate-based Digital Signature Algorithm) is a signature mechanism with verification key $Y = [X^{-1}]G$; that is, the parameter D is equal to −1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness function is defined by the formula

$$R = h(FE2BS(r, \Pi_X)).$$

If γ is longer than β, then the witness function is replaced by the formula

$$R = I2BS(\beta', BS2I(\gamma, h(FE2BS(r, \Pi_X))) \bmod 2^{\beta'})$$

where β' is $8 * \lceil \beta / 8 \rceil$.

Domain parameters shall indicate the employed hash-function. The assignment function is defined by the formula

$$(T_1, T_2) = (V, -1)$$

where $V = BS2I(\beta, (R \oplus H)) \bmod q$. The value H is the hash-code from public key Y and message M.

The coefficients (A, B, C) of the EC-KCDSA signature formula are set as follows

$$(A, B, C) = (T_2, S, T_1).$$

Thus the signature formula becomes

$$-K + SX^{-1} + V \equiv 0 \pmod q.$$

NOTE This mechanism is taken from Reference [37]. The notation here has been changed slightly from Reference [37] to conform with notation used elsewhere in this part of ISO/IEC 14888.

6.7.2 Parameters

l	the input block size (in bits) of the selected hash-function h
$\boldsymbol{F}$	a finite field
$\mathbb{E}$	an elliptic curve group over field $\boldsymbol{F}$
$\#\mathbb{E}$	the cardinality of $\mathbb{E}$
q	a prime divisor of $\#\mathbb{E}$
G	a point on the elliptic curve of order q

Hash-function identifier or OID with specified hash-function

All these parameters can be public and can be common to a group of users.

It is recommended that all users check the proper generation of the EC-KCDSA public parameters according to Reference [37].

6.7.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is –1. The corresponding public verification key Y is

$$Y = [X^{-1}]G.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.7.4 Signature process

6.7.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.7.4.2 Producing the pre-signature

The input to this stage is the randomizer K and the signing entity computes

$$\Pi = [K]G.$$

6.7.4.3 Preparing the message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.7.4.4 Computing the witness

The signing entity computes $R = h(FE2BS(r, \Pi_X))$, where output of h is the hash-code of Π_X. If γ is longer than β, then the computation of witness is replaced by $R = I2BS(\beta', BS2I(\gamma, h(FE2BS(r, \prod_X))) \bmod 2^{\beta'})$.

The conversion rules, $FE2BS$, $I2BS$ and $BS2I$, are given in Annex B.

6.7.4.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (V, -1)$ where $V = BS2I(\beta', R \oplus H) \bmod q$, where $H=h(Y' \,||\, M_2)$ is the hash-code of the concatenation of Y' and message M_2. The value of Y' is the leftmost l bits of a bit sequence $FE2BS(r, Y_X)||FE2BS(r, Y_Y)$. If l is longer than the length of the sequence, 0 is padded as much as needed after the sequence. In computing V, the bit string $R \oplus H$ shall be converted to an integer before modulo reduction with respect to q.

If γ is longer than β, then the computation of H is replaced by $H = I2BS(\beta', BS2I(\gamma, h(Y' \,||\, M_2)) \bmod 2^{\beta'})$.

NOTE Y' is a fixed value for a user, thus, this value can be kept as a user parameter.

6.7.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed in 6.7.4.4 and

$$S = X(K - V) \bmod q.$$

6.7.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.7.4.8 Constructing the signed message

A signed message is the concatenation of a message, M, and the appendix.

$M \,||\, ((R, S), text)$

6.7.5 Verification process

6.7.5.1 General

Prior to verifying the signature of a signed message, it is necessary that the verifier has trusted copies of E, q and G.

The verifier also acquires the necessary data items for the verification process; for example, the verification key Y (see ISO/IEC 14888-1:2008, Clause 9 for additional required data items).

6.7.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then checks whether the following conditions hold or not:

— $0 < S < q$;

— if the length of the value γ is not longer than β, the bit length of R is equal to the output bit length of the employed hash-function h;

— if the length of the value γ is longer than β, the bit length of R is equal to β.

If any of the above condition does not hold the signature shall be rejected.

6.7.5.3 Preparing the message for verification

The verifier retrieves $M_2 = M$ from the signed message. M_1 is empty.

6.7.5.4 Retrieving the assignment

This stage is identical to 6.7.4.5. The inputs to the assignment function consist of the witness R from 6.7.5.2 and M_2 from 6.7.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.7.4.5.

6.7.5.5 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.7.5.4 and the second part of the signature S from 6.7.5.2. The verifier obtains a recomputed value Π' of the pre-signature using the formula

$$\Pi' = [S \bmod q]Y + [T_1 \bmod q]G.$$

6.7.5.6 Recomputing the witness

The computations at this stage are the same as in 6.7.4.4. The verifier executes the witness function. The input is Π' from 6.7.5.5. Note that M_1 is empty. The output is the recomputed witness R'.

6.7.5.7 Verifying the witness

The verifier compares the recomputed witness R' from 6.7.5.6 to the value of R from 6.7.5.2. If $R' = R$, then the signature is valid.

6.8 EC-GDSA

6.8.1 General

EC-GDSA (Elliptic Curve German Digital Signature Algorithm) is a signature mechanism with verification key $Y = [X^{-1}]G$; that is, the parameter D is equal to −1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The witness function is defined by the formula

$$R = FE2I(r, \Pi_X) \bmod q.$$

The coefficients (A, B, C) of the EC-GDSA signature formula are set as follows:

$$(A, B, C) = (T_1, S, T_2)$$

where $(T_1, T_2) = (-R, H)$ and H is the hash-code of the message M.

Thus the signature formula becomes

$$-RK + SX^{-1} + H \equiv 0 \pmod q.$$

NOTE EC-GDSA stands for Elliptic Curve German Digital Signature Algorithm.[16]

6.8.2 Parameters

F	a finite field
E	an elliptic curve group over field ***F***
$\#E$	the cardinality of E
q	a prime divisor of $\#E$
G	a point on the elliptic curve of order q

Hash-function identifier or OID with specified hash-function

All these parameters can be public and can be common to a group of users.

It is recommended that all users check the proper generation of the public parameters.

6.8.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is –1. Thus, the corresponding public verification key Y is

$$Y = [X^{-1}]G.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.8.4 Signature process

6.8.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.8.4.2 Producing the pre-signature

The input to this stage is the randomizer K and the signing entity computes

$$\Pi = [K]G.$$

6.8.4.3 Preparing message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.8.4.4 Computing the witness

The signing entity computes $R = FE2I(r, \Pi_X) \bmod q$ where the witness is simply a function of the pre-signature.

6.8.4.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (-R, BS2I(\gamma, H))$ where H is the hash-code of the message M_2.

6.8.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed in 6.8.4.4 and

$$S = X(KR - BS2I(\gamma, H)) \bmod q.$$

6.8.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*, $((R, S), text)$.

6.8.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix

$$M||((R, S), text).$$

6.8.5 Verification process

6.8.5.1 General

The verifying entity acquires the necessary data items required for the verification process.

6.8.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then first checks to see that $0 < R < q$ and $0 < S < q$; if either condition is violated the signature shall be rejected.

6.8.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2. M_1 will be empty and $M_2 = M$.

6.8.5.4 Retrieving the assignment

This stage is identical to 6.8.4.5. The inputs to the assignment function consist of the witness R from 6.8.5.2 and M_2 from 6.8.5.3. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.8.4.5.

6.8.5.5 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.8.5.4 and second part of the signature S from 6.8.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = [(-T_1)^{-1} S \bmod q]Y + [(-T_1)^{-1} T_2 \bmod q]G.$$

6.8.5.6 Recomputing the witness

The computations at this stage are the same as in 6.8.4.4. The verifier executes the witness function. The input is Π' from 6.8.5.5. The output is the recomputed witnessR'.

6.8.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.8.5.6 to the retrieved version of R from 6.8.5.2. If $R' = R$, then the signature is verified.

6.9 EC-RDSA

6.9.1 General

EC-RDSA (Elliptic Curve Russian Digital Signature Algorithm) is a signature mechanism with verification key $Y = [X]G$; that is, the parameter D is equal to 1. The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$. The coefficients (A, B, C) of the EC-RDSA signature formula are set as follows:

$$(A, B, C) = (T_1, T_2, -S)$$

where $(T_1, T_2) = (H, R)$ and $H = h(M)$ is the hash-code of message M, converted to an integer as described in 6.9.4.5.

The witness function is defined by the formula

$$\mathrm{R} = \mathrm{FE2I}(\mathrm{r}, \Pi_X) \bmod \mathrm{q}.$$

Thus the signature formula becomes

$$HK + RX - S \equiv 0 \pmod{q}.$$

NOTE EC-RDSA stands for Elliptic Curve Russian Digital Signature Algorithm. The mechanism is taken from Reference [21]. The notation here has been changed from Reference [22] to conform with the notation used in ISO/IEC 14888.

6.9.2 Parameters

p	a prime
E	an elliptic curve group over the field $GF(p)$
$\#E$	the cardinality of E
q	a prime divisor of $\#E$
G	a point on the elliptic curve of order q

Hash-function identifier or OID with specified hash-function

All these parameters can be public and can be common to a group of users.

6.9.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is

$$Y = [X]G.$$

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

NOTE Reference [21] does not completely specify the process of generation of a user's secret signature key X.

6.9.4 Signature process

6.9.4.1 Producing the randomizer

The signing entity generates a random or pseudo-random integer K such that $0 < K < q$.

6.9.4.2 Producing the pre-signature

The input to this stage is the randomizer K, and the signing entity computes

$$\Pi = [K]G.$$

6.9.4.3 Preparing message for signing

The message is prepared such that M_1 is empty and M_2 is the message to be signed, i.e. $M_2 = M$.

6.9.4.4 Computing the witness

The signing entity computes $R = FE2I(r, \Pi_X) \bmod q$.

6.9.4.5 Computing the assignment

The signing entity computes $H = h(M_2)$. H is then converted to an integer according to conversion rule $BS2I$ in Annex B. If H is equal to 0 mod q, then H is set to 1. The assignment (T_1, T_2) is $(BS2I(\gamma, H), R)$, if $BS2I(\gamma, H) \neq 0 \pmod q$, or $(1, R)$ otherwise.

6.9.4.6 Computing the second part of the signature

The signature is (R, S) where R is computed as given in 6.9.4.4 and

$$S = RX + KH \bmod q.$$

The signer should check whether $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated.

6.9.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field *text*, i.e. it will equal $((R, S) \| \mathit{text})$.

6.9.4.8 Constructing the signed message

A signed message is the concatenation of the message M and the appendix

$$\mathrm{M} \| ((R, S) \| \mathit{text}).$$

6.9.5 Verification process

6.9.5.1 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier then checks whether $0 < R < q$ and $0 < S < q$; if either condition does not hold, the signature shall be rejected.

6.9.5.2 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2. M_1 will be empty and $M_2 = M$.

6.9.5.3 Retrieving the assignment

This stage is identical to 6.9.4.5. The inputs to the assignment function consist of the witness R from 6.9.5.1 and M_2 from 6.9.5.2. The assignment $T = (T_1, T_2)$ is recomputed as the output from the assignment function given in 6.9.4.5.

6.9.5.4 Recomputing the pre-signature

The inputs to this stage are system parameters, the verification key Y, the assignment $T = (T_1, T_2)$ from 6.9.5.3, and the second part of the signature S from 6.9.5.1. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = [-T_1^{-1}T_2 \bmod q]Y + [T_1^{-1}S \bmod q]G.$$

6.9.5.5 Recomputing the witness

The computations at this stage are the same as in 6.9.4.4. The verifier executes the witness function. The input is Π' from 6.9.5.4. The output is the recomputed witness, R'.

6.9.5.6 Verifying the witness

The verifier compares the recomputed witness, R' from 6.9.5.5 to the retrieved version of R from 6.9.5.1. If $R' = R$, then the signature is verified.

6.10 EC-SDSA

6.10.1 General

EC-SDSA (Elliptic Curve Schnorr Digital Signature Algorithm) is a signature mechanism with verification key $Y = [X]G$; that is, the parameter D is equal to 1. The message is prepared such that $M2$ is empty and $M1 = M$ the message to be signed. The witness R is computed as a hash-code of the message M and a random pre-signature $\prod = [K]G$, by one of two methods, either

normal $R = h(FE2BS(r, \Pi_X) \,||\, FE2BS(r, \Pi_Y) \,||\, M)$

or

optimized $R = h(FE2BS(r, \Pi_X) \,||\, M)$.

The first method generates the witness by hashing the concatenation of the x-coordinate of Π, the y-coordinate of Π and the message M. The second method omits the y-coordinate from the hash calculation and thereby improves performance.

The second method is an optimized variant of EC-SDSA (see Reference [30]).

The coefficients (A, B, C) of the EC-SDSA signature formula are set as follows:

$$(A, B, C) = (T_1, T_2, S)$$

where $(T_1, T_2) = (-1, -BS2I(\gamma, R) \bmod q)$.

Thus the signature formula becomes

$$-K + T_2X + S \equiv 0 \pmod q).$$

NOTE EC-SDSA stands for Elliptic Curve Schnorr Digital Signature Algorithm. The mechanism is taken from Reference [33]. The notation here has been changed from Reference [33] to conform with the notation used in ISO/IEC 14888.

6.10.2 Parameters

$\boldsymbol{F}$	a finite field
E	elliptic curve group over the field $\boldsymbol{F}$
$\#E$	cardinality of E
q	prime divisor of $\#E$
G	a point of order q on the elliptic curve

Hash-function identifier or OID with specified hash-function

All these parameters can be public and can be common to a group of users.

It is recommended that all users check the proper generation of the public parameters according to Reference [7].

6.10.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is $Y = [X]G$.

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.10.4 Signature process

6.10.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.10.4.2 Producing the pre-signature

The input to this stage is the randomizer K, and the signing entity computes $\Pi = [K]G$.

6.10.4.3 Preparing message for signing

The message is prepared such that M_1 is the message to be signed, i.e. $M_1 = M$, and M_2 is empty.

6.10.4.4 Computing the witness

The signing entity computes $R = h(FE2BS(r, \Pi_X)||FE2BS(r, \Pi_Y)||M)$.

For the optimized variant of EC-SDSA, the signing entity instead computes $R = h(FE2BS(r, \Pi_X)||M)$.

6.10.4.5 Computing the assignment

The value of the witness R is converted to an integer according to conversion rule, *BS2I,* in Annex B and then reducing modulo q.

The assignment $(T1, T2)$ is $(-1, -BS2I(\gamma, R) \bmod q)$.

6.10.4.6 Computing the second part of the signature

The signature is (R, S) where $S = (K + BS2I(\gamma, R)X) \bmod q$.

As an option, one may wish to check if $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated (it is extremely unlikely that $R = 0$ or $S = 0$ if signatures are generated properly).

6.10.4.7 Constructing the appendix

The appendix will be the concatenation of (R, S) and an optional text field, *text*.

6.10.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix

$M \,||\, ((R, S) \,||\, text)$.

6.10.5 Verification process

6.10.5.1 General

The verifying entity acquires the necessary data items required for the verification process.

6.10.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier first checks to see that R is a non-zero string within the range of the hash function and $0 < S < q$; if either condition is violated the signature shall be rejected.

6.10.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2, such that $M_1 = M$ and M_2 is empty.

6.10.5.4 Retrieving the assignment

The input to the assignment function consists of the witness R from 6.10.5.2. The assignment $T = (T_1, T_2) = (-1, -BS2I(\gamma, R) \bmod q)$.

6.10.5.5 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.10.5.4 and second part of the signature S from 6.10.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = [-ST_1 \bmod q]G + [T_2 \bmod q]Y = [S \bmod q]G + [T_2 \bmod q]Y.$$

6.10.5.6 Recomputing the witness

The computations at this stage are the same as in 6.10.4.4 and 6.10.4.5. The verifier executes the witness function from 6.10.4.4. The input is Π' from 6.10.5.5. The output is the recomputed witness R' which is the hash-code of the recomputed pre-signature Π' and the message M.

6.10.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.10.5.6 to the retrieved version of R from 6.10.5.2. If $R' = R$, then the signature is verified.

6.11 EC-FSDSA

6.11.1 General

EC-FSDSA (Elliptic Curve Full Schnorr Digital Signature Algorithm) is a signature mechanism with verification key $Y = [X]G$; that is, the parameter D is equal to 1. The message is prepared such that M_1 is empty and $M_2 = M$ the message to be signed. The witness R is computed as

$$R = FE2BS(r, \Pi_X)||FE2BS(r, \Pi_Y).$$

The coefficients (A, B, C) of the EC-FSDSA signature formula are set as follows:

$$(A, B, C) = (T_1, T_2, S)$$

where $T = (T_1, T_2) = (-1, -BS2I(\gamma, h(R||M)) \bmod q)$.

Thus, the signature formula becomes

$$-K + T_2X + S \equiv 0 \pmod q.$$

NOTE EC-FSDSA stands for Elliptic Curve Full Schnorr Digital Signature Algorithm. The mechanism is taken from Reference [33]. The notation here has been changed from Reference [33] to conform with the notation used in ISO/IEC 14888.

6.11.2 Parameters

$\boldsymbol{F}$	a finite field
E	an elliptic curve group over the field $\boldsymbol{F}$
$\#E$	the cardinality of E
q	a prime divisor of $\#E$
G	a point of order q on the elliptic curve E

Hash function identifier or OID with specified hash function

All these parameters can be public and can be common to a group of users.

It is recommended that all users check the proper generation of the public parameters according to Reference [7].

6.11.3 Generation of signature key and verification key

The signature key of a signing entity is a secretly generated random or pseudo-random integer X such that $0 < X < q$. The parameter D is 1. The corresponding public verification key Y is $Y = [X]G$.

A user's secret signature key X and public verification key Y are normally fixed for a period of time. The signature key X shall be kept secret.

6.11.4 Signature process

6.11.4.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0 < K < q$.

6.11.4.2 Producing the pre-signature

The input to this stage is the randomizer K, and the signing entity computes $\Pi = [K]G$.

6.11.4.3 Preparing message for signing

The message is prepared such that M is the message to be signed, i.e. $M_2 = M$, and M_1 is empty.

6.11.4.4 Computing the witness

The signing entity computes $R = FE2BS(r, \Pi_X)||FE2BS(r, \Pi_Y)$.

6.11.4.5 Computing the assignment

The signing entity computes the hash code $h(R||M)$. Afterwards, the hash code is converted to an integer according to conversion rule, $BS2I$, in Annex B and then reduced modulo q. The assignment (T_1, T_2) is $(-1, -BS2I(\gamma, h(R||M)) \bmod q)$.

6.11.4.6 Computing the second part of the signature

The signature is (R, S) where $S = (K+BS2I(\gamma, h(R||M))X \bmod q)$.

As an option, one may wish to check if $R = 0$ or $S = 0$. If either $R = 0$ or $S = 0$, a new value of K should be generated and the signature should be recalculated. (It is extremely unlikely that $R = 0$ or $S = 0$ if signatures are generated properly).

6.11.4.7 Constructing the appendix

The appendix will be the concatenation of (R,S) and an optional text field, *text*.

6.11.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix:

$$(M \,||\, (R,S) \,||\, text).$$

6.11.5 Verification process

6.11.5.1 General

The verifying entity acquires the necessary data items required for the verification process.

6.11.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix. The verifier first checks to see that R represents a point on E and $0 < S < q$; if either condition is violated the signature shall be rejected.

6.11.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2, such that $M_2 = M$ and M_1 is empty.

6.11.5.4 Retrieving the assignment

The input to the assignment function is computed as in 6.11.4.5 from the witness R from 6.11.4.4 and the message M from 6.11.4.3. The assignment is given by $T = (T_1, T_2) = (-1, -BS2I(\gamma, h(R||M)) \bmod q)$.

6.11.5.5 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1, T_2)$ from 6.11.5.4 and second part of the signature S from 6.11.5.2. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$$\Pi' = [-ST_1 \bmod q]G + [T_2 \bmod q]Y = [S \bmod q]G + [T_2 \bmod q]Y.$$

6.11.5.6 Recomputing the witness

The computations at this stage are the same as in 6.11.4.4. The verifier executes the witness function. The input is Π' from 6.11.5.5. The output is the recomputed witness R'.

6.11.5.7 Verifying the witness

The verifier compares the recomputed witness, R' from 6.11.5.6 to the retrieved version of R from 6.11.5.2. If $R' = R$, then the signature is verified.

7 Identity-based mechanisms

7.1 General

7.1 7.1 General

The following data items are required for the signature process:

— domain parameters, E, $GF(r)$, G_1, G_2, q, P, $< >$;

— public master key V;

— signature key X;

— message M;

— hash-function identifier for H_1 and H_2 (optional);

— identity string ID;

— other text (optional).

The hash-function identifier can be used for binding the signature mechanism and the hash-function.

The following data items are required for the verification process:

— domain parameters, E, $GF(r)$, G_1, G_2, q, P, $< >$;

— public master key V;

— verification key Y, which can be derived from an identity string;

— message M;

— signature $\sum$;

— hash-function identifier for H_1 and H_2 (optional);

NOTE 1 The signer and verifier have to agree on the specific hash-functions for h, H_1 and H_2 used in the mechanism. If any of these hash-functions are not uniquely determined by other means, the hash-function identifier is required in both the signature and verification processes (see ISO/IEC 14888-1).

— Identity string ID;

— other text (optional).

NOTE 2 Typical elliptic curves for IBS-1 and IBS-2 are super-singular elliptic curves over $GF(r)$, where $r = p^m$, p is a prime ≥2 and m is an integer ≥1.

7.2 IBS-1

7.2.1 General

IBS-1 is an identity-based signature scheme on an additive group of elliptic curve points. It takes

$$(A, B, C) = (T_1, S, T_2)$$

where $T_1 = -Y$, $T_2 = [R]Y$. $D = -1$. Thus, the signature formula becomes

$$[-K]Y + [U^{-1}]S + [R]Y \equiv 0_E \text{ (in } G_1).$$

NOTE This mechanism is based on the algorithm designed by Hess in Reference [22].

7.2.2 Parameters

The signature mechanism takes place in an environment where the entities involved share the following parameters, which have been defined in Clause 4: G_1, G_2, P, q, $< >$, H_1, and H_2.

It is recommended that all users check the proper generation of the public parameters.

7.2.3 Generation of master key and signature/verification key

A master key pair of a KGC is (U, V), where U is the master private key generated by choosing an integer such that $0 < U < q$ at random, and V is the master public key generated by computing $V = [U]P$. The KGC publishes V and keeps U secret.

A signature and verification key pair of a signer is (X, Y), where Y is the public verification key generated from an identity string ID and a hash-function H_1, i.e. $Y = H_1(ID)$, and X is the private signature key generated by computing $X = [U]Y$, which is done by the KGC and given to the signer.

7.2.4 Signature process

7.2.4.1 Producing the randomizer

The signer first chooses a random or pseudo-random integer K such that $0 < K < q$. The signer keeps the value K secret.

7.2.4.2 Producing the pre-signature

The signer takes K, P and X as input to produce the pre-signature result

$$\Pi = <X, P>^K.$$

NOTE Π is an element in an extension field of $GF(p^m)$ and the extension has degree 4 for characteristic $p = 2$, degree 6 for characteristic $p = 3$ and degree 2 for characteristic $p > 3$.

7.2.4.3 Preparing message for signing

The signer prepares the message such that M_2 is empty and M_1 is the signed message M, i.e. $M_1 = M$.

7.2.4.4 Computing the witness

Let $\Pi = (\Pi_a, \Pi_b)$. The signer applies the hash-function H_2 to $M_1||FE2BS(r, \Pi_a)||FE2BS(r, \Pi_b)$ (concatenation of M_1, $FE2BS(r, \Pi_a)$ and $FE2BS(r, \Pi_b)$) to obtain the witness

$R = BS2I(\gamma, H_2(M_1 || FE2BS(r, \Pi_a) || FE2BS(r, \Pi_b))) \bmod q$.

If $R = 0$, output "invalid" and stop.

For fields of higher extension degree, more terms will appear in the value to be hashed. For example, for extension degree 3, $\Pi = (\Pi_a, \Pi_b, \Pi_c)$ and the input to H_2 would be

$M_1 \,||\, FE2BS(r, \Pi_a) \,||\, FE2BS(r, \Pi_b) \,||\, FE2BS(r, \Pi_c)$.

7.2.4.5 Computing the assignment

The assignment is $T = (T_1, T_2)$ as $(-Y, [R]Y)$. However, it is not necessary for the signer to compute the assignment.

7.2.4.6 Computing the second part of the signature

The signer computes the second part of the signature as

$$S = [K - R]X.$$

The signature is $\Sigma = (R, S)$.

7.2.4.7 Constructing the appendix

The signer constructs the appendix that is the concatenation of (R, S) and an optional text field, *text*, i.e. $((R, S), text)$.

7.2.4.8 Constructing the signed message

The signer constructs the signed message that is the concatenation of the message, M, and the appendix, i.e. $M||((R, S), text)$.

7.2.5 Verification process

7.2.5.1 General

The verifier first acquires the necessary data items required for the verification process.

7.2.5.2 Retrieving the witness

The verifier retrieves the witness R and the second part of the signature S from the appendix.

The verifier then checks if $S \in G_1$ holds; if the condition is violated, the signature shall be rejected. Otherwise, the verifier carries out the following steps.

7.2.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2 such that M_2 is empty and M_1 is equal to M.

7.2.5.4 Retrieving the assignment

The assignment is $T = (T_1, T_2)$, where $T_1 = -Y$, and $T_2 = [R]Y$. However, it is not necessary for the verifier to compute the assignment.

7.2.5.5 Recomputing the pre-signature

The verifier recomputes the pre-signature value

$\Pi' = <S, P> * <Y, V>^R$.

NOTE The pairing $<Y, V>$ can be pre-computed.

7.2.5.6 Recomputing the witness

The verifier recomputes the witness

$R' = \mathrm{BS2I}(\gamma, H_2(M_1 || \mathrm{FE2BS}(r, \Pi_a') || \mathrm{FE2BS}(r, \Pi_b'))) \bmod q$.

For fields of higher extension degree, more terms will appear in the value to be hashed. For example, for extension degree 3, $\Pi' = \Pi_a', \Pi_b', \Pi_c'$) and the input to H_2 would be

$M_1 || \mathrm{FE2BS}(r, \Pi_a') || \mathrm{FE2BS}(r, \Pi_b') || \mathrm{FE2BS}(r, \Pi_c')$.

7.2.5.7 Verifying the witness

The verifier checks whether $R' = R$ holds. If it holds, the signature is verified; otherwise, it is invalid.

7.3 IBS-2

7.3.1 General

IBS-2 is an identity-based signature scheme on an additive group of elliptic curve points. It takes

$(A, B, C) = (T_1, S, T_2)$

where $T_1 = -Y$, $T_2 = [-H]Y$, and H is a hash-code from H_2. $D = -1$.

Thus the signature formula becomes

$[-K]Y + [U^{-1}]S + [-H]Y \equiv 0_E$ (in G_1).

NOTE This mechanism is based on the algorithm designed by Cha and Cheon in Reference [15].

7.3.2 Parameters

The parameters are the same as in 7.2.2.

7.3.3 Generation of master key and signature/verification key

This operation is the same as in 7.2.3.

7.3.4 Signature process

7.3.4.1 Producing the randomizer

The signer first chooses a random or pseudo-random integer K such that $0 < K < q$. The signer keeps the value K secret.

7.3.4.2 Producing the pre-signature

The signer takes K and Y as input to produce the pre-signature result

$$\Pi = [K]Y.$$

7.3.4.3 Preparing message for signing

The signer prepares the message such that M_1 is empty and M_2 is the signed message M, i.e. $M_2 = M$.

7.3.4.4 Computing the witness

The signer obtain the witness from the pre-signature result

$$R = \Pi .$$

7.3.4.5 Computing the assignment

The assignment is $T = (T_1, T_2)$ as

$T_1 = -Y$, and

$T_2 = [-H]Y$

where $H = BS2I(\gamma, H_2(M_2||FE2BS(r, \Pi_X)))$ mod q. However, the signer only needs to compute the value of H.

7.3.4.6 Computing the second part of the signature

The signer computes the second part of the signature as

$$S = [K + H]X.$$

The signature is $\Sigma = (R, S)$.

7.3.4.7 Constructing the appendix

The signer constructs the appendix that is the concatenation of (R, S) and an optional text field, *text*, i.e. $((R, S)$, *text*).

7.3.4.8 Constructing the signed message

The signer constructs the signed message that is the concatenation of the message, M, and the appendix, i.e. $M||((R, S)$, *text*).

7.3.5 Verification process

7.3.5.1 General

The verifier first acquires the necessary data items required for the verification process.

7.3.5.2 Retrieving the witness

The verifier retrieves the pre-signature R and the second part of the signature S from the appendix.

The verifier first checks if R and $S \in G_1$ holds; if either condition is violated the signature shall be rejected. Otherwise, the verifier carries out the following steps.

7.3.5.3 Preparing message for verification

The verifier retrieves M from the signed message and divides the message into two parts M_1 and M_2 such that M_1 is empty and M_2 is equal to M.

7.3.5.4 Retrieving the assignment

The assignment is $T = (T_1, T_2)$ as

$T_1 = -Y$, and

$T_2 = [-H]Y$.

where $H = BS2I(\gamma, H_2(M_2||FE2BS(r, \prod_X)))$ mod q. However, the verifier only needs to recompute the value of H.

7.3.5.5 Recomputing the pre-signature

The verifier retrieves the pre-signature value as

$\Pi' = R$.

7.3.5.6 Recomputing the witness

Instead of recomputing the witness value R, the verifier computes two pairings $<P, S>$ and $<V, \Pi' + [H]Y>$.

7.3.5.7 Verifying the witness

The verifier checks if $<P, S> = <V, \Pi' + [H]Y>$ holds. If it holds, then the signature is verified; otherwise, it is invalid.

Annex A
(normative)

Object identifier

Annex A lists the object identifiers assigned to the digital signature mechanisms specified in this part of ISO/IEC 14888, and defines algorithm parameter structures.

```
DigitalSignatureWithAppendixDL {
   iso(1) standard(0) digital-signature-with-appendix (14888) part3(3)
      asn1-module(1) discrete-logarithm-based-mechanisms(0) }
DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

IMPORTS

   HashFunctions
      FROM DedicatedHashFunctions {
         iso(1) standard(0) encryption-algorithms(10118) part3(3)
         asn1-module(1)
             dedicated-hash-functions(0) } ;

OID ::= OBJECT IDENTIFIER -- alias

-- Synonyms --

id-dswa-dl OID ::= {
   iso(1) standard(0) digital-signature-with-appendix(14888) part3(3)
      algorithm(0) }

-- Assignments --

id-dswa-dl-DSA      OID  ::=   {   iso(1)  member-body(2)  us(840)   ansi-x9-57(10040)
x9cm(4) dsa(1) }
id-dswa-dl-KCDSA    OID  ::=   { id-dswa-dl kcdsa(2) }
id-dswa-dl-PVS      OID  ::=   { id-dswa-dl pvs(3) }
id-dswa-dl-EC-DSA   OID  ::=   {   iso(1)  member-body(2)  us(840)   ansi-x9-62(10045)
signatures(4) ecdsa-with-Recommended(2) }
id-dswa-dl-EC-KCDSA OID  ::=   { id-dswa-dl ec-kcdsa(5) }
id-dswa-dl-EC-GDSA  OID  ::=   { id-dswa-dl ec-gdsa(6) }
id-dswa-dl-IBS-1    OID  ::=   { id-dswa-dl ibs-1(7) }
id-dswa-dl-IBS-2    OID  ::=   { id-dswa-dl ibs-2(8) }
id-dswa-dl-EC-RDSA  OID  ::=   { id-dswa-dl ec-rdsa(9) }
id-dswa-dl-SDSA     OID  ::=   { id-dswa-dl sdsa(10) }
id-dswa-dl-EC-SDSA  OID  ::=   { id-dswa-dl ec-sdsa(11) }
id-dswa-dl-EC-FSDSA OID  ::=   { id-dswa-dl ec-fsdsa(12) }
id-dswa-dl-EC-SDSA-opt OID ::= { id-dswa-dl ec-sdsa-opt(13) }

DigitalSignatureWithAppendix ::= SEQUENCE {
   algorithm  ALGORITHM.&id({DSAlgorithms}),
   parameters ALGORITHM.&Type({DSAlgorithms}{@algorithm})  OPTIONAL
}

DSAlgorithms ALGORITHM ::= {
   dswa-dl-DSA        |
   dswa-dl-KCDSA      |
   dswa-dl-PVS        |
   dswa-dl-EC-DSA     |
   dswa-dl-EC-KCDSA |
   dswa-dl-EC-GDSA    |
   dswa-dl-IBS-1      |
   dswa-dl-IBS-2      |
   dswa-dl EC-RDSA    |
   dswa-dl SDSA       |
   dswa-dl EC-SDSA  |
   dswa-dl EC-FSDSA |
   dswa-dl EC-SDSA-opt,

   ...  -- Expect additional algorithms --
}

dswa-dl-DSA ALGORITHM ::= {
   OID id-dswa-dl-DSA PARMS NullParms
}

dswa-dl-KCDSA ALGORITHM ::= {
   OID id-dswa-dl-KCDSA PARMS HashFunctions
}

dswa-dl-PVS ALGORITHM ::= {
   OID id-dswa-dl-PVS PARMS HashFunctions
}

dswa-dl-EC-DSA ALGORITHM ::= {
   OID id-dswa-dl-EC-DSA PARMS NullParms
}

dswa-dl-EC-KCDSA ALGORITHM ::= {
   OID id-dswa-dl-EC-KCDSA PARMS HashFunctions
}

dswa-dl-EC-GDSA ALGORITHM ::= {
   OID id-dswa-dl-EC-GDSA PARMS HashFunctions
}

dswa-dl-IBS-1 ALGORITHM ::= {
   OID id-dswa-dl-IBS-1 PARMS HashFunctions
}

dswa-dl-IBS-2 ALGORITHM ::= {
   OID id-dswa-dl-IBS-2 PARMS HashFunctions
}

dswa-dl-EC-RDSA ALGORITHM ::= {
   OID id-dswa-dl-EC-RDSA PARMS HashFunctions
}

dswa-dl-SDSA ALGORITHM ::= {
OID id-dswa-dl-SDSA PARMS HashFunctions
}

dswa-dl-EC-SDSA ALGORITHM ::= {
OID id-dswa-dl-EC-SDSA PARMS HashFunctions
}

dswa-dl-EC-FSDSA ALGORITHM ::= {
OID id-dswa-dl-EC-FSDSA PARMS HashFunctions
}

dswa-dl-EC-SDSA-opt ALGORITHM ::= {
   OID id-dswa-dl-EC-SDSA-opt PARMS HashFunctions
}


NullParms ::= NULL

-- Cryptographic algorithm identification --

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

END -- DigitalSignatureWithAppendixDL --
```

NOTE 1 Alternative OIDs for KCDSA presented in KCAC.TG.OID are as follows:

```
 {iso(1) member-body(2) korea(410) kisa(20004) npki-alg(1) kcdsa1(21)}
```
- KCDSA

```
{iso(1) member-body(2) korea(410) kisa(20004) npki-alg(1) kcdsa1WithHAS160(22)}
```
- KCDSA with HAS160, where the HAS160 is a Korean hash standard algorithm

```
{iso(1) member-body(2) korea(410) kisa(20004) npki-alg(1) kcdsa1WithSHA1(23)}
```
- KCDSA with SHA1

NOTE 2 Alternative OID for EC-KCDSA with HAS160 presented in TTAS.KO-12.0015 is

```
 {iso(1) member-body(2) korea(410) kisa(20004) npki-alg(1) ecc(100) signature(4)
    eckcdsa-with-HAS160(1)}.
```

Annex B
(normative)

Conversion functions (I)

B.1 Conversion from a field element to an integer: *FE2I*(*r*, *x*)

Input values

— r – a prime or a power of a prime.

— x – an element of the Galois field $GF(r)$.

Assumptions

— When $r = p$, where p is an odd prime:

$x \in GF(p)$ is (already) represented as an integer in $\{0, 1, \ldots, p - 1\}$.

— When $r = p^m$, where p is an odd prime and m is an integer greater than 1:

$x \in GF(p^m)$ is represented as a p-ary string of length m;

$x = x_{m-1}\, x_{m-2} \ldots x_0$, where $x_i \in \{0, 1, \ldots, p - 1\}$ for $0 \leq i < m$.

— When $r = 2$:

$x \in GF(2)$ is (already) represented as an integer in $\{0, 1\}$.

— When $r = 2^m$, where m is an integer greater than 1:

$x \in GF(2^m)$ is represented as a binary string of length m;

$x = x_{m-1}\, x_{m-2} \ldots x_0$, where $x_i \in \{0, 1\}$ for $0 \leq i < m$.

Output value

— When $r = p$, where p is an odd prime:

$FE2I(r, x) = x \in \{0, 1, \ldots, p - 1\}$.

— When $r = p^m$, where p is an odd prime, m is an integer greater than 1, and x is represented as the p-ary string $x_{m-1}\, x_{m-2} \ldots x_0$:

$FE2I(r, x) = p^{m-1} x_{m-1} + p^{m-2} x_{m-2} + \ldots + x_0 \in \{0, 1, \ldots, p^m - 1\}$.

— When $r = 2$:

$FE2I(r, x) = x \in \{0, 1\}$.

— When $r = 2^m$, where m is an integer greater than 1 and x is represented as the binary string x_{m-1} $x_{m-2} \ldots x_0$:

$FE2I(r, x) = 2^{m-1} x_{m-1} + 2^{m-2} x_{m-2} + \ldots + x_0 \in \{0, 1, \ldots, 2^m - 1\}$.

B.2 Conversion from an integer to a field element: *I2FE*(r, x)

Input values

— r – a prime or a power of a prime.

— x – an integer in {0, 1, ..., $r - 1$}.

Assumptions

— When $r = p$, where p is an odd prime:

Elements of $GF(p)$ are represented as integers in $\{0, 1, \ldots, p - 1\}$.

— When $r = p^m$, where p is an odd prime and m is an integer greater than 1:

Elements of $GF(p^m)$ are represented as a p-ary strings of length m.

— When $r = 2$:

Elements of $GF(2)$ are represented integers in {0, 1}.

— When $r = 2^m$, where m is an integer greater than 1:

Elements of $GF(2^m)$ are represented as a binary strings of length m.

Output value

— When $r = p$ is an odd prime:

$I2FE(r, x) = x \in \{0, 1, \ldots, p - 1\}$.

— When $r = p^m$, where p is an odd prime and m is an integer greater than 1:

$I2FE(r, x) = x_{m-1}\, x_{m-2} \ldots x_0$.

A p-ary string whose m components (representing a base p expansion of x, padded with leading zeros, if necessary, to attain the desired length) can be computed as follows.

```
$a := x$;
i := 0;
While ( $i < m$ ) do {
          $b := \lfloor a/p \rfloor$;
          $x_i := a - (p)(b)$;
          $a := b$;
          $i := i + 1$ }
```

— When $r = 2$:

$I2FE(r, x) = x \in \{0, 1\}$.

— When $r = 2^m$, where m is an integer greater than 1:

$I2FE(r, x) = x_{m-1}\, x_{m-2} \ldots x_0$,

a binary string whose m components (representing a base 2 expansion of x, padded with leading zeros, if necessary, to attain the desired length) can be computed as follows.

```
$a := x;$
$i := 0;$
While ( $i < m$ ) do {
        $b := \lfloor a/2 \rfloor;$
        $x_i := a - (2)(b);$
        $a := b;$
        $i := i + 1$ }
```

B.3 Conversion from a field element to a binary string: *FE2BS*(*r*, *x*)

Input values

— r – a prime or a power of a prime.

— x – an element of the Galois field $GF(r)$.

Assumptions

— When $r = p$, where p is an odd prime:

$x \in GF(p)$ is represented as an integer in $\{0, 1, \ldots, p - 1\}$.

— When $r = p^m$, where p is an odd prime and m is an integer greater than 1:

$x \in GF(p^m)$ is represented as a p-ary string of length m;

$x = x_{m-1}\, x_{m-2} \ldots x_0$, where $x_i \in \{0, 1, \ldots, p - 1\}$ for $0 \le i < m$.

— When $r = 2$:

$x \in GF(2)$ is represented as an integer in $\{0, 1\}$.

— When $r = 2^m$, where m is an integer greater than 1:

$x \in GF(2^m)$ is represented as a binary string of length m;

$x = x_{m-1}\, x_{m-2} \ldots x_0$, where $x_i \in \{0, 1\}$ for $0 \le i < m$.

Output value

— $FE2BS(r, x) = I2BS(\, g, FE2I(r, x)\,)$,

where $g = 8 \lceil \log_{256}(\, r\,) \rceil$.

B.4 Conversion from a binary string to an integer: *BS2I* (*g*, *x*)

Input values

— g – a positive integer, indicating the length of the input string.

— $x = x_{g-1}\, x_{g-2} \ldots x_0$ – a binary string of length g.

Output value

— $BS2I(g, x) = 2^{g-1} x_{g-1} + 2^{g-2} x_{g-2} + \ldots + x_0 \in \{0, 1, \ldots, 2^g - 1\}$.

B.5 Conversion from an integer to a binary string: *I2BS*(*g*, *x*)

Input values

— g – a positive integer, indicating the length of the output string.

— x – an integer in $\{0, 1, ..., 2^g - 1\}$.

Output value

— $I2BS(g, x) = x_{g-1}\, x_{g-2} \ldots x_0$.

A binary string whose g components (representing a base 2 expansion of x, padded with leading zeros, if necessary, to attain the desired length) can be computed as follows.

```
$a := x$;
i := 0;
While ( $i < g$ ) do {
        $b := \lfloor a/2 \rfloor$;
        $x_i := a - (2)(b)$;
        $a := b$;
        $i := i + 1$ }
```

B.6 Conversion between an integer and an octet string: *I2OS*(*h*, *x*) & *OS2I*(*h*, *M*)

***I2OS*(*h*, *x*):**

Input values

— h – a positive integer, indicating the length of the output octet string.

— x – an integer in $\{0, 1, ..., 256^h - 1\}$.

Output value

— Compute a string of integers, $x_{h-1}\, x_{h-2} \ldots x_0$, where $x_i \in \{0, 1, \ldots, 255\}$ for $0 \le i < h$, representing a base 256 expansion of x, padded with leading zeros, if necessary, to attain length, h. The x_i values can be computed as follows.

```
$a := x$;
i := 0;
While ( $i < h$ ) do {
        $b := \lfloor a/256 \rfloor$;
        $x_i := a - (256)(b)$;
        $a := b$;
        $i := i + 1$ }
```

— $I2OS(h, x) = M_{h-1}\, M_{h-2} \ldots M_0$,

where octet M_i is equivalent to the 8-long binary string $I2BS(8, x_i)$.

***OS2I*(*h*, *M*):**

Input values

— h – a positive integer, indicating the length of the input octet string.

— $M = M_{h-1}\, M_{h-2} \ldots M_0$ – an octet string of length h.

Assumption

— For $0 \leq i < h$, M_i is represented as an 8-long binary string.

Output value

— Compute the string of integers, $x_{h-1}\, x_{h-2} \ldots x_0$,

where $x_i = BS2I(8, M_i) \in \{0, 1, \ldots, 255\}$ for $0 \leq i < h$.

$OS2I(h, x) = 256^{h-1} x_{h-1} + 256^{h-2} x_{h-2} + \ldots + x_0 \in \{0, 1, \ldots, 256^h - 1\}$.

Annex C
(informative)

Conversion functions (II)

Annex C specifies Function *I2P* (Integer to Point conversion), which is used to specify the two identity based signature mechanisms in Clause 7.

This function is believed to hold the properties of the randomness and onewayness, i.e. converting an integer to a point such that the point is randomly distributed in the selected group and that given current knowledge, recovering the integer from the point is computationally infeasible. This function is also used by IEEE P1363. However, there is no formal security proof of this function published. For this reason, this function is recommended as informative.

Given a set of elliptic curve domain parameters (r, q, a_1, a_2), Function *I2P* operates on an integer u as input, and produces a point T of order q on the curve E over $GF(r)$ as output, which is specified as $T = I2P(u)$. In the following specification, the operations of addition and multiplication between finite field elements follow the specification in ISO/IEC 15946-1, and the operation of KDF1 follows the specification in ISO/IEC 18033-2.

a) Set $v = BS2I(8\ \lceil \log_{256}(r) \rceil, KDF1_{H2}\ (I2OS(\lceil \log_{256}(\ u\ ) \rceil, u)$, length in bytes of representation of r)) mod r. If $v = 0$, output "invalid" and stop.

b) Set $\lambda = u$ mod 2.

c) If r is a prime ($r = p$) and the curve E is $Y^2 = X^3 + a_1X + a_2$ defined over $GF(p)$, compute the point T in the following way:

   1) Let x have the value of v.

   2) Compute the field element $c = x^3 + a_1x + a_2$ mod p. If $c = 0$, output "invalid" and stop.

   3) Find a square root d of c modulo p (i.e. an integer d with $0 < d < p$ such that $d^2 = c$ mod p) or determine that no such square roots exist.

      i) To determine the existence of the square root, compute $\delta = c^{(p-1)/2}$ mod p. If $\delta = 1$, d exists, otherwise, d does not exist.

      ii) If $\delta \neq 1$, compute $u = u + 1$ and go to Step a).

      iii) If $\delta = 1$, find d.

      The operation of finding two field elements d such that $d^2 = c$ mod p is given in Reference [4] and Reference [23]. In order to make the result unique, if the application of this mechanism has a specific requirement on which value should be chosen, follow the requirement. Otherwise, a smallest absolute value modulo p is recommended.

      iv) Set $y = (I2FE(r, p - 1))^{\lambda} \times d$.

      v) Set point $T = (x, y)$, compute $T=[\#E/q]T$, and output T.

d) If r is an odd prime power ($r = p^m$, $p > 2$, $m \geq 2$) and the curve E is $Y^2 = x^3 + a_1x^2 + a_2$ (when $p = 3$) and $Y^2 = x^3 + a_1x + a_2$ (when $p > 3$) defined over $GF(p^m)$, compute the point T in the following way:

   1) Set $x = I2FE(r, v)$.

   2) If ($p = 3$), set $c = x^3 + a_1x^2 + a_2$ in $GF(p^m)$. If $c = 0$, output "invalid" and stop.

3) If ($p > 3$), set $c = x^3 + a_1x + a_2$ in $GF(p^m)$. If $c = 0$, output "invalid" and stop.

4) Find a square root d of c in $GF(p^m)$ [i.e. a $GF(p^m)$ element d such that $d^2 = c$ in $GF(p^m)$] or determine that no such square roots exist. If the result is no such square roots existing, Set $u = u + 1$ and go to Step a).

NOTE 1 The operations of determining the existence of and finding a square root of a field element are given in References [4] and [23]. In order to make the result unique, if the application of this mechanism has a specific requirement on which value should be chosen, follow the requirement. Otherwise, compare the maximum degrees of the results, and select d with smaller degree. If the maximum degree of the values are same, then select d with smaller absolute value coefficient of the degree. If both the degree and the coefficient are same, then compare the second largest degree and select d with smaller absolute value coefficient of the degree. Repeat this process until the unique d is selected.

5) Set $y = (I2FE(r, p - 1))^{\lambda} \times d$.

6) Set point $T = (x, y)$, compute $T = [\#E/q]T$, and output T.

e) If r is a prime power of 2 ($r = 2^m$, $m \geq 2$) and the curve E is $Y^2 + XY = X^3 + a_1X^2 + a_2$ defined over $GF(2^m)$, compute the point T in the following way:

1) Set $x = I2FE(r, v)$.

2) Set $c = x + a_1 + a_2x^{(-2)}$ in $GF(2^m)$. If $c = 0$, output "invalid" and stop.

3) Find a field element d satisfying $d^2 + d \equiv c$ in $GF(2^m)$ or determine that no such integers exist. If the result is no such integers existing, Set $u = u + 1$ and go to Step a).

NOTE 2 The operations of determining the existence of and finding a field element d such that $d^2 + d = c$ in $GF(2^m)$ is given in References [4] and [23]. In order to make the result unique, if the application of this mechanism has a specific requirement on which value should be chosen, follow the requirement. Otherwise, compare the maximum degrees of the results, and select d with smaller maximum degree. If the maximum degree of two values are same, then compare the second largest degree and select d with smaller second largest degree. Repeat this process until the unique d is selected.

4) Set $y = (d + I2FE(r, \lambda)) \times x$.

5) Set point $T = (x, y)$, compute T=[$\#E/q$]T, and output T.

Annex D
(normative)

Generation of DSA domain parameters

D.1 Generation of the prime *p* and *q*

The prime generation scheme starts by using the appropriate hash-function and a user supplied *SEED* to construct a prime q, in the range $2^{\beta-1} < q < 2^{\beta}$. Once this is accomplished, the same *SEED* value is used to construct an X in the range $2^{\alpha-1} < X < 2^{\alpha}$. The prime p is then formed by rounding X to a number congruent to 1 mod $2q$ as described below. Conversion functions between integer and sequence are given in Annex B.

Let h be the appropriate hash-function for the (α, β) pair, and let m (= γ) be the length of its output block in bits. Let α-1 = $n*m + b$, where b and n are integers, and $0 \leq b < m$.

Step 1. Choose an arbitrary sequence of at least β bits and call it *SEED*. Let s be the length of *SEED* in bits.

Step 2. Compute $U = h(SEED) \bmod 2^{\beta}$.

Step 3. Form q from U by setting the most significant bit (the $2^{\beta - 1}$ bit) and the least significant bit to 1. In terms of Boolean operations, $q = U$ OR $2^{\beta - 1}$ OR 1. Note that $2^{\beta - 1} < q < 2^{\beta}$.

Step 4. Use a robust primality testing algorithm to test whether q is prime (a robust primality test is one where the probability of a non-prime number passing the test is at most $2^{-\beta/2}$).

Step 5. If q is not a prime, go to Step 1.

Step 6. Let *counter* = 0 and *offset* = 1.

Step 7. For $k = 0, \ldots, n$ let

$Vk = h((SEED + offset + k) \bmod 2^{s})$.

Step 8. Let W be the integer $W = V0 + V1 * 2^{m} + \ldots + Vn\text{-}1 * 2^{(n-1)*m} + (Vn \bmod 2^{b}) * 2^{n*m}$ and let $X = W + 2^{\alpha-1}$. Note that $0 \leq W < 2^{\alpha-1}$ and hence $2^{\alpha-1} \leq X < 2^{\alpha}$.

Step 9. Let $c = X \bmod 2q$ and set $p = X - (c - 1)$. Note that p is congruent to 1 mod $2q$.

Step 10. If $p < 2^{\alpha-1}$, then go to Step 13.

Step 11. Perform a robust primality test on p.

Step 12. If p passes the test performed in Step 11, go to Step 15.

Step 13. Let *counter* = *counter* + 1 and *offset* = *offset* + n + 1.

Step 14. If *counter* $\geq 4\alpha$ go to Step 1, otherwise (i.e. if *counter* $< 4\alpha$) go to Step 7.

Step 15. Save the value of counter and optionally the value of *SEED* for use in certifying the proper generation of p and q.

NOTE This procedure is taken from Reference [17], Appendix A.

D.2 Generation of the generator *G*

D.2.1 Unverifiable generation of *G*

This method is used to determine the generator G when the validation of G is not required. The value of G is determined from p and q.

Step 1. $e = (p\text{-}1)/q$.

Step 2. Set F = any integer, where $1 < F < p\text{-}1$, and F differs from any value previously tried.

Step 3. $G = F^{e} \bmod p$.

Step 4. If $G = 1$, then go to Step 2.

D.2.2 Verifiable generation of *G*

For this method, the generator G is based on the values of p, q, *index* and *SEED*. The *index* is a bit string of length eight that represents an unsigned integer. *index* can be used to produce different values of G from the same (p, q) pair. The value of *SEED* is the final saved value from the algorithm described in D.1. Let h be the appropriate hash-function for the (α, β) pair. Note that this method supports the generation of multiple values of G for specific values of p and q. The use of different values of G may be used to support key separation by providing different values for *index*.

Here, “ggen” is an ACSII byte string 0x6767656E, and *count* is a 16-bit counter (i.e. an unsigned integer defined module 2^{16}).

Step 1. $e = (p\text{-}1)/q$.

Step 2. $count = 1$.

Step 3. $U = SEED \,||\, \text{“ggen”} \,||\, index \,||\, count$.

Step 4. $W = h(U)$.

Step 5. $G = W^{e} \bmod p$.

Step 6. If $G < 2$, then increment *count* and go to Step 3.

Annex E
(informative)

The Weil and Tate pairings

E.1 General

The Weil pairing and Tate pairing are both functions $<P, Q>$ of pairs P, Q of points on an elliptic curve E. They are used in the two identity-based mechanisms specified in Clause 7.

Let G_1 and G_2 denote two groups of prime order q, where G_1, with an additive notation, denotes the group of points on an elliptic curve E; and G_2, with a multiplicative notation, denotes a subgroup of the multiplicative group of a finite field.

A pairing is a computable bilinear map between these two groups. Two pairings have been studied for cryptographic use. They are Weil pairing[27][35] and a modified version[11] and a modified version of Tate pairing.[18][19] In this part of ISO/IEC 14888, < > denotes a general bilinear map, i.e. $< >: G_1 \times G_1 \rightarrow G_2$, which can be either a modified Weil pairing or Tate pairing.

The modified Weil pairing and Tate pairing have the following two properties:

— Bilinear: If P, P_1, P_2, Q, Q_1, Q_2 are points of a cyclic group of prime order q and a satisfying $1 \leq a \leq q - 1$;

$<P_1 + P_2, Q> = <P_1, Q> * <P_2, Q>$;

$<P, Q_1 + Q_2> = <P, Q_1> * <P, Q_2>$;

$<[a]P, Q> = <P, [a]Q> = <P, Q>^a$;

— Non-degenerate: If P is a non-identity point of the cyclic group, $<P, P> \neq 1$.

E.2 The functions *f*, *g* and *d*

The following three functions are used to compute the Weil and Tate pairings.

— Let E be an elliptic curve with the formula $y^2 + a_1*x*y + a_3*y = x^3 + a_2*x^2 + a_4*x + a_6$.

— Given three finite points (x_0, y_0), (x_1, y_1), (u, v) on E, define the function $f((x_0, y_0), (x_1, y_1), (u, v))$ by

if $(x_0, y_0) = 0_E$ and $(x_1, y_1) = 0_E$	then	$f = 1$
else if $(x_0, y_0) = 0_E$	then	$f = u - x_1$
else if $(x_1, y_1) = 0_E$	then	$f = u - x_0$
else if $x_0 \neq x_1$	then	$f = (x_1 - x_0) * v - (y_1 - y_0) * u - x_1*y_0 + x_0 * y_1$
else if $y_0 \neq y_1$	then	$f = u - x_0$
Else		$f = (a_1 * y_0 - 3* x_0^2 - 2*a_2* x_0 - a_4) * (u - x_0) + (2* y_0 + a_1* x_0 + a_3) * (v - y_0)$ $= - (v - y_0)^2 - (u - x_0) * (a_1* (v - y_0) - (u - x_0) * (2 * x_0 + a_2 + u))$

— Given points A, B, C on E, let g(A, B, C) = f(A, B, C) / f(A + B, – (A + B), C).

NOTE Dependent on the value *r* defined in Clause 4 and the curve *E*, the above function *f* might be simplified. The following are a few widely used examples, which do not cover every possible case. If *r* is a prime ($r = p$) and the curve *E* is $y^2 = x^3 + a_4x + a_6$ defined over $GF(p)$, or if *r* is an odd prime power ($r = p^m$, $p > 3$, $m \geq 2$) and the curve *E* is $y^2 = x^3 + a_4{*}x + a_6$ defined over $GF(p^m)$, the function *f* can be written as $f = (-3{*}x_0^2 - a_4) * (u - x_0) + 2 {*}y_0{*} (v - y_0) = - (v - y_0)^2 + (u - x_0)^2{*} (2{*}x_0 + u)$. If *r* is a prime power of 2 ($r = 2^m$, $m \geq 2$) and the curve *E* is $y^2 + x{*}y = x^3 + a_2{*}x^2 + a_6$ defined over $GF(2^m)$, the function *f* can be written as $f = (y_0 + x_0^2) * (u + x_0) + x_0{*} (v + y_0) = (v + y_0^2) + (u + x_0) * (v + y_0 + (u + x_0) * (u + a_2))$. If *r* is a power of 3 ($r = 3^m$, $m \geq 2$) and the curve *E* is $Y^2 = x^3 + a_2x^2 + a_6$ defined over $GF(3^m)$, the function *f* can be written as $f = 2{*}y_0{*}(v - y_0) - 2{*}a_2{*}x_0 * (u - x_0)$.

— Given two points D and C on E and one integer l > 2, the Weil function d(D, C, l) is computed via the following algorithm.

a) Set $A = D$, $f = 1$. Let $l = (n_t, ..., n_0)$ be the bit representation of *l* such that $l = \sum_i n_i 2^i$ and $n_t \neq 0$.

b) for $i = t - 1, t - 2, ..., 0$ do {

```
        $f = f * f * g(A, A, C)$;
        $A = A + A$;
        if $n_i \neq 0$ then {
            $f = f * g(A, D, C)$;
            $A = A + D$;
        }
    }
```

c) Set $d(D, C, l) = f$ and output $d(D, C, l)$.

Omitting the parameter *l*, $d(D, C, l)$ is denoted as $d(D, C)$.

E.3 The Weil pairing

Let $l > 2$ be prime, and let *P* and *Q* be points on *E* with $[l]P = [l]Q = 0_E$, the Weil pairing $< P, Q >$ can be computed in the following steps:

— choose some random point *T* on *E* (such that 0_E, *Q*, *T*, $P + T$ are all distinct), then

— compute $<P, Q> = ((d(P, Q - T) / d(P, -T)) / (d(Q, P + T) / d(Q, T)))$.

If during the computation of the pairing, a division by zero is attempted, then the computation should be restarted with a new point *T*.

E.4 The Tate pairing

Let $l > 2$ be prime, and let *P* and *Q* be points on *E* with $[l]P = 0_E$, the Tate pairing $< P, Q >$ can be computed in the following steps:

— choose some random point *T* on *E*, then

— compute $<P, Q> = (d(P, Q - T) / d(P, -T))$.

If during the computation of the pairing, a division by zero is attempted, then the computation should be restarted with a new point *T*.

NOTE More detailed information of the Weil and Tate pairing implementation can be found in References [8], [21] and [30].

E.5 The reduced Tate pairing

Let $l > 2$ be prime, and let P and Q be points on E with $[l]\ P = O$, the pairing $<P, Q>$ can be computed in the following steps:

— choose some random point T on E;

— compute $<P, Q> = (\ d(P, Q - T)\ /\ d(P, -T)\ )^{(pk - 1)\ /\ l}$;

where k is the dimension of the extension field (e.g. $k = 2$) and p^k is the number of elements in the extension field. If during the computation of the pairing, a division by zero is attempted, then the computation should be restarted with a new point T.

NOTE 1 More detailed information of pairing implementation can be found in References [4] and [23].

NOTE 2 The reduced Tate pairing is used in numerical examples of F.11 and F.12.

Annex F
(informative)

Numerical examples

F.1 General

In Annex F, unless otherwise explicitly stated, all the values are expressed in hexadecimal notation.

It is recommended that digital signatures based on SHA-1 and RIPEMD-160 should only be used for legacy applications.

F.2 DSA mechanism

F.2.1 Example 1: 2048-bit Prime P, SHA-224

F.2.1.1 General

A complete explanation of the generation of all values is given in Reference [17]. This example is sample value for DSA with α = 2048 and β = 224. All hashing, including generation of domain parameters, is performed with SHA-224.

F.2.1.2 Parameters

α = 2048

β = 224

SEED = 0C088E11 2F88B186 90421876 5614496E C2AF9770 C71D0A56 87F489B6

F = 2

P = B4865EFC 44BFB4CB 7EE034F0 EAE8A72D 25897819 9BF9BA28 8462FD97
19F33272 C010A11B 33BCE4E8 481B6EC7 AB1229D9 FC7BEA43 8055907F
F1E28FAC 33716089 DCED277F 9036440A 887D4B22 CAC5BABD ECD6A1B3
A1731594 20371025 BAAB5F18 D5FDE928 CE4F5EE4 5352785F 20057782
2C20756E 171CBDD8 1CEB932A E0F29109 5CFFD9C2 3A07AC6B C2F5250B
B9F8E2E6 5AF85215 6E8EEBF8 31C098FB 010057BD 425132B8 0A46BB5C
E801E241 05058E58 091383F1 6F124894 FB6DE9CD 3BCC4C6E 64901743
AF8F47C3 5CC2177E B15ED172 B4969174 FE3F645A 9D3BEFC6 811A9074
BF702024 98E5E157 ECDBED3C 1FDF3C4F 00DAB43A CBA49802 79392E18
B515851F

Q = B4D0963C 40D74138 69F42710 BBEF73CB C6C1C4E6 35C6B9F3 CF7A6255

```
Counter = 24

      G = A92434D5 6752B028 CF11954E 0F3B1BED 8804EB74 8DEED793 E2932E80
          8F37C34A 15444A06 9A8B17E5 4BF7FB82 7D6FE959 428BA0CC 1F3B2B8E
          EA0A25A2 CAF73A0C 68C7DC48 093374A3 CD1F2250 8EF05038 9E8AE58C
          E6A8AD50 2510B4CA C42528B7 BCA0993C C959C630 61D7BA3A 885E9C6D
          CA6EAE44 E2D3C050 A236645F FBDE4BA6 1ECEB17B 941F85E9 C5234A28
          FAD461DE 8B55F033 DB7E0CB4 DA5E115F FFCD416D 5A8BC9CD 9DAA6816
          010841CC 9F416A6F E109A40A 823874F0 EDD92F45 738918AC 0CB925E7
          AB8E692A 9336DB36 697E6C75 5B0243CA EBB61A38 79EABAE6 AC53F166
          2740D6ED 3E3DB9BF A629390A 6A517FB0 B50D02E2 57178145 AF964626
          57ABA465
```

F.2.1.3 Signature key and verification key

```
      X = A279D0A3 A4243A2B 16909C9E 0BBFEC32 0589E4DF 1BDDAE72 3BA7353B
      Y = 31246FA1 CB8D1430 BDCDEBF0 5BB8C967 D24E6728 BA5C900C 50852741
          3AFD496A F12EA9CC D80D8916 62A7B9B3 C2023212 08943D85 5D7EA110
          B9512D1B 9E4AABAB 72B99005 25127129 EAB2CC8E 66B6E09C 49341ABF
          184B2733 9114E39E FED6B90B 8D7BA182 3E3512D3 EB82F720 76C2815D
          A642DE61 D808DCF0 22A76077 1E22AA42 26997E41 EA142BAD BFD00011
          F7D27677 08A0313E 42255286 0D184F18 C4890ED3 A6CE8134 E1647DDC
          B292B5FD 5C5ED61C 1BF9567A E1E40CC5 F85F5B7D 1A09AAA1 08CFCFE2
          469360A9 48F61B4D 1CDCA791 1BB64070 94D9A78B A34ED943 97057791
          DFC56691 1B4F7DD9 61A7EBB8 74923C59 2458D43D F171CB81 698AB7EE
          2E9B92E6
```

F.2.1.4 Per message data

```
     M= ASCII form of "abc" = 61 62 63
     K= 2973C724 7F9BD6DB 3C08CD7A 1DA427DF 6780A7DD F3E09362 E8BA1293
  h(M)= 23097D22 3405D822 8642A477 BDA255B3 2AADBCE4 BDA0B3F7 E36C9DA7
```

F.2.1.5 Signature

```
      R = 1DFAAA6F 87DA6148 6529A2F3 4EBC7D89 3D42F405 F8DCBB33 93CC1A00
     S= 4A3E6377 D09A4CD6 67BA9F9C E3982EB9 C1AA6E90 70F7C2F7 0EA23173
```

F.2.1.6 Verification

R' = 1DFAAA6F 87DA6148 6529A2F3 4EBC7D89 3D42F405 F8DCBB33 93CC1A00

F.2.2 Example 2: 3072-bit Prime P, SHA-256

F.2.2.1 General

This example is sample value for DSA with $\alpha = 3072$ and $\beta = 256$. All hashing, including generation of domain parameters, is performed with SHA-256.

F.2.2.2 Parameters

α = 3072

β = 256

SEED = 193AFCA7 C1E77B3C 1ECC618C 81322E47 B8B8B997 C9C83515 C59CC446
C2D9BD47

F = 2

P = 90066455 B5CFC38F 9CAA4A48 B4281F29 2C260FEE F01FD610 37E56258
A7795A1C 7AD46076 982CE6BB 956936C6 AB4DCFE0 5E678458 6940CA54
4B9B2140 E1EB523F 009D20A7 E7880E4E 5BFA690F 1B9004A2 7811CD99
04AF7042 0EEFD6EA 11EF7DA1 29F58835 FF56B89F AA637BC9 AC2EFAAB
90340222 9F491D8D 3485261C D068699B 6BA58A1D DBBEF6DB 51E8FE34
E8A78E54 2D7BA351 C21EA8D8 F1D29F5D 5D159394 87E27F44 16B0CA63
2C59EFD1 B1EB6651 1A5A0FBF 615B766C 5862D0BD 8A3FE7A0 E0DA0FB2
FE1FCB19 E8F9996A 8EA0FCCD E5381752 38FC8B0E E6F29AF7 F642773E
BE8CD540 2415A014 51A84047 6B2FCEB0 E388D30D 4B376C37 FE401C2A
202F941D AD179C54 0C1C8CE0 30D460C4 D983BE9A B0B20F69 144C1AE1
3F9383EA 1C08504F B0BF3215 03EFE434 88310DD8 DC77EC5B 8349B8BF
E97C2C56 0EA878DE 87C11E3D 597F1FEA 742D73EE C7F37BE4 3949EF1A
0D15C3F3 E3FC0A83 35617055 AC91328E C22B50FC 15B941D3 D1624CD8
8BC25F3E 941FDDC6 20068958 1BFEC416 B4B2CB73

Q = CFA0478A 54717B08 CE64805B 76E5B142 49A77A48 38469DF7 F7DC987E
FCCFB11D

$Counter$ = 20

G = 5E5CBA99 2E0A680D 885EB903 AEA78E4A 45A46910 3D448EDE 3B7ACCC5
4D521E37 F84A4BDD 5B06B097 0CC2D2BB B715F7B8 2846F9A0 C393914C

```
792E6A92 3E2117AB 805276A9 75AADB52 61D91673 EA9AAFFE ECBFA618
3DFCB5D3 B7332AA1 9275AFA1 F8EC0B60 FB6F66CC 23AE4870 791D5982
AAD1AA94 85FD8F4A 60126FEB 2CF05DB8 A7F0F09B 3397F393 7F2E90B9
E5B9C9B6 EFEF642B C48351C4 6FB171B9 BFA9EF17 A961CE96 C7E7A7CC
3D3D03DF AD1078BA 21DA4251 98F07D24 81622BCE 45969D9C 4D6063D7
2AB7A0F0 8B2F49A7 CC6AF335 E08C4720 E31476B6 7299E231 F8BD90B3
9AC3AE3B E0C6B6CA CEF8289A 2E2873D5 8E51E029 CAFBD55E 6841489A
B66B5B4B 9BA6E2F7 84660896 AFF387D9 2844CCB8 B6947549 6DE19DA2
E58259B0 90489AC8 E62363CD F82CFD8E F2A427AB CD65750B 506F56DD
E3B98856 7A88126B 914D7828 E2B63A6D 7ED0747E C59E0E0A 23CE7D8A
74C1D2C2 A7AFB6A2 9799620F 00E11C33 787F7DED 3B30E1A2 2D09F1FB
DA1ABBBF BF25CAE0 5A13F812 E34563F9 9410E73B
```

F.2.2.3 Signature key and verification key

```
X = 3ABC1587 297CE7B9 EA1AD665 1CF2BC4D 7F92ED25 CABC8553 F567D1B4
  = 0EBB8764
Y = 8B891C86 92D3DE87 5879390F 2698B26F BECCA6B0 75535DCE 6B0C8625
    77F9FA0D EF6074E7 A7624121 224A5958 96ABD4CD A56B2CEF B942E025
    D2A4282F FAA98A48 CDB47E1A 6FCB5CFB 393EF35A F9DF9131 02BB303C
    2B5C36C3 F8FC04ED 7B8B69FE FE0CF3E1 FC05CFA7 13B3435B 2656E913
    BA8874AE A9F93600 6AEB448B CD005D18 EC3562A3 3D04CF25 C8D3D698
    44343442 FA3DB7DE 618C5E2D A064573E 61E6D558 1BFB694A 23AC87FD
    5B52D62E 954E1376 DB8DDB52 4FFC0D46 9DF97879 2EE44173 8E5DB05A
    7DC43E94 C11A2E7A 4FBE3830 71FA36D2 A7EC8A93 88FE1C4F 79888A99
    D3B61056 97C2556B 79BB4D7E 781CEBB3 D4866AD8 25A5E830 84607228
    9FDBC941 FA679CA8 2F5F78B7 461B2404 DB883D21 5F4E0676 CF549395
    0AC55916 97BFEA8D 1EE6EC01 6B89BA51 CAFB5F9C 84C989FA 117375E9
    4578F28B 3ABC1587 297CE7B9 E0B34CE0 545DA462 66FD77F6 2D8F2CEE
    92AB7701 2AFEBC11 008985A8 21CD2D97 8C7E6FE7 499D1AAF 8DE632C2
    1BB48CA5 CBF9F310 98FD3FD3 854C49A6 5D920174 4AACE540 354974F9
```

F.2.2.4 Per message data

M = ASCII form of "abc" = `61 62 63`

K = `A6902C1E 6E3943C5 62806158 8A8B007B CCEA91DB F1291548 3F04B24A B0678BEE`

$h(M)$ = `BA7816BF 8F01CFEA 414140DE 5DAE2223 B00361A3 96177A9C B410FF61 F20015AD`

F.2.2.5 Signature

R = `5F184E64 5A38BE8F B4A6871B 6503A9D1 2924C7AB E04B7141 0066C2EC A6E3BE3E`

S = `91EB0C7B A3D4B9B6 0B825C3D 9F2CADA8 A2C9D772 3267B033 CBCDCF88 03DB9C18`

F.2.2.6 Verification

R' = `5F184E64 5A38BE8F B4A6871B 6503A9D1 2924C7AB E04B7141 0066C2EC A6E3BE3E`

F.3 KCDSA mechanism

F.3.1 Example 1: 2048-bit Prime P, 224-bit Prime Q, SHA-224

F.3.1.1 General

This example uses SHA-224 as the hash-function h. The hash-code is simply the value of SHA-224.

F.3.1.2 Parameters

l = `200` (i.e. 512 in decimal)

α = `2048`

β = `224`

P =
```
8DA8C1B5 C95D11BE 46661DF5 8C9F803E B729B800 DD92751B
3A4F10C6 A5448E9F 3BC0E916 F042E399 B34AF9BE E582CCFC
3FF5000C FF235694 94351CFE A5529EA3 47DCF43F 302F5894
380709EA 2E1C416B 51A5CDFC 7593B18B 7E3788D5 1B9CC9AE
828B4F8F B06E0E90 57F7FA0F 93BB0397 031FE7D5 0A6828DA
0C1160A0 E66D4E5D 2A18AD17 A811E70B 14F4F431 1A028260
3233444F 98763C5A 1E829C76 4CF36ADB 56980BD4 C54BBE29
```

```
      7E790228 4292D75C A3600FF4 59310B09 291CBEFB C721528A
      13403B8B 93B711C3 03A2182B 6E6397E0 83380BF2 886AF3B9
      AFCC9F50 55D8B713 6C0EBD08 C5CF0B38 888CD115 72787F6D
      F384C97C 91B58C31 DEE5655E CBF3FA53
  Q = 864F1884 1EC103CD FD1BE7FE E54650F2 2A3BB997 537F32CC
      79A51F53
  G = 0E9BE1F8 7A414D16 7A9A5A96 8B079E4A D385A357 3EDB21AA
      67A6F61C 0D00C14A 7A225044 B6E9EB03 68C1EB57 B24B45CD
      854FD93C 1B2DFB0A 3EA302D2 367E4EC7 2F6E7EE8 EA7F8002
      F7704E99 0B954F25 BADA8DA6 2BAEB6F0 6953C0C8 5104AD03
      F36618F7 6C62F4EC F3480183 69850A56 17C999DB E68BA17D
      5BC72556 74EF4839 22C6A3F9 9D3C3C6F 358896C4 E63C605E
      E7DB16FC BD9BE354 E281F7FE 7813D054 27ED1912 B5C7653A
      167B9434 9147EEAF 85CC9CE2 E81661F3 21512D5D 2C0580B0
      3D1704EE F2317F45 185C8258 387E7EC9 79C04707 EF546241
      2784AFE4 1A7B45C8 3B9CBE48 F9127CB4 400BE9E9 6AC5DE17
      F2C9DEA3 5E3734E7 9B64673F 85681C4E
```

F.3.1.3 Signature key and verification key

```
  X = 2F1991C1 AF401872 8A5A431B 9B5459DF B16F6D25 6797FE57
      0EC6BC65
  Y = 04EDE5C6 7EA29297 A8CACB6B DE6F4666 AEA27D10 3DD1E9E9
      582F76A2 F22B8B1B 32230BC5 8F06B768 F8102B49 FA1CAE5E
      18921494 7F6239B6 C6CE7C9B C2D230E8 9A40BEE2 C33A8861
      FD4F7D35 B788FE95 B2D5885D 8C8FAEA8 1C90BE4C EE2784E3
      3577A71D 3B7F085D 71E9A1D4 7815C73F A087ACAA B9FCB565
      5AC9570E 6852BE7C 9C0AECEA 8BD9AA75 A44FC314 7F733E90
      6ADB0FD7 6D613561 B1DB364B BDC9AFD3 CE8F5F17 E3E71203
      4A999350 8059FA52 441FA90D DFE9A0F2 A0B9192F E2220C08
      1BD0C0F0 E07CB5F1 EE4FF405 23591F17 8A4FC7CB 5065F6A3
      8216E9A0 99C205B2 9B8746D8 65E1AF6D 903E5A13 8004910B
      70EB5B84 EED9760E A60578BF 08852898
```

F.3.1.4 Per message data

M = ASCII form of "This is a test message for KCDSA usage!" =

```
54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6D
65 73 73 61 67 65 20 66 6F 72 20 4B 43 44 53 41
20 75 73 61 67 65 21
```

K = `49561994 FD2BAD5E 410CA1C1 5C3FD3F1 2E70263F 2820AD5C`
`566DED80`

Y' = `1BD0C0F0 E07CB5F1 EE4FF405 23591F17 8A4FC7CB 5065F6A3`
`8216E9A0 99C205B2 9B8746D8 65E1AF6D 903E5A13 8004910B`
`70EB5B84 EED9760E A60578BF 08852898`

$h(Y' || M)$ = `B3F921C1 8DDD06B6 09D02BD5 916F180E 5DFB19C8 9FEC063D`
`059A3575`

V = `5E4E4BEC B42ED14C 1F00A98C D077A8C1 D65E6F5A 50D7ACD1`
`6AF7EC24`

F.3.1.5 Signature

R = `EDB76A2D 39F3D7FA 16D08259 4118B0CF 8BA57692 CF3BAAEC`
`6F6DD951`

S = `5260A2DF 2E923DE8 77B130AC 8B5E8B17 63973B88 D5D4627A`
`DFBACF52`

F.3.1.6 Verification

R' = `EDB76A2D 39F3D7FA 16D08259 4118B0CF 8BA57692 CF3BAAEC`
`6F6DD951`

F.3.2 Example 2: 3072-bit Prime P, 256-bit Prime Q, SHA-256

F.3.2.1 General

This example uses SHA-256 as the hash-function h. The hash-code is simply the value of SHA-256.

F.3.2.2 Parameters

l = `200` (i.e. 512 in decimal)

α = `3072`

β = `256`

P = `CBAEACE3 677E98AD B2E49C00 2B8B0F43 4143B466 515839BF`

```
      813B097D 2D1EE681 5008C27A 3415BC22 31609874 5E5844F3
      3ECC8887 C16DFB1C FB77DC4C 3F3571CC EEFD4291 8F6C48C3
      702AB6EF 0919B7E8 402FC89B 35D09A0E 5040E309 1EE4674B
      E891933C 1007E017 EDD40818 7E4114B6 BE5548D7 8DB58B84
      8475A422 62D7EB79 5F08D161 1055EFEA 8A6AEB20 EB0F1C22
      F002A2E8 195BCBBA 830B8461 3531BDD9 EC71E5A9 7A9DCCC6
      5D6117B8 5D0CA66C 3FDAA347 6E97ADCD 05A1F490 2BD04B92
      F400C42B A0C9940A 32600443 3B6D3001 28BF930F 484EAA63
      02CD7A31 9EE5E561 A12A3625 594020C2 40DBA3BE BD8A4751
      5841F198 EBE43218 2639616F 6A7F9BD7 434F0534 8F7E1DB3
      115A9FEE BA984A2B 73784334 DE7737EE 3704535F CA2F4904
      CB4AD58F 172F2648 E1D62D05 8539AC78 3D032D18 33D2B9AA
      D96982C9 692E0DDB B6615508 83ED66F7 AA8BCE8F F0663A0A
      DDA226C7 BD0E06DF C72594A3 87C676A3 CA06A300 62BE1D85
      F23E3E02 C4D65E06 1B619B04 E83A318E C55ECA06 9EB85603
  Q = C2A8CAF4 87180079 66F2EC13 4EABA3CB B07F31A8 F2667ACB
      5D9B872F A760A401
  G = 17A1C167 AF836CC8 5149BE43 63F1BB4F 0010848F C9B678B4
      E026F1F3 87133749 A4B1BBA4 C23252A4 C86F31E2 1E8ACACB
      4E33AD89 B7C3D79A 5409268B FBA82B45 814E4352 0C09D631
      613FA35D B9CAF18F 791C2729 A4B014BC 79A85A90 CD541037
      119ECCDE 0778863F FCB9C259 31FCD33A 6706E5FE 1F495BB8
      BCB3D0EE C9B6D5A9 373127A2 121E37D9 8A840330 258DBFCE
      E7E06F81 5B69C16C 5D17289C 4CC37E71 9B856298 D4E1574E
      4F4F8515 BAF9A850 D11DDA09 55BC30FA 5B16792D 673A3B1F
      41512FC3 EB89452D 51509F97 4D878B48 2D2AD2ED 32BE1905
      6F574504 2BFF804F B7482796 612B746F E8D70A83 8CC6F496
      DD0FFC3D 95C1E0B1 98184D73 523656A0 6431BC52 5C2BC161
      9729E8C0 88F6DF91 5645E060 922A4AF3 EDD63047 C7B6077C
      667C07D8 8EB00F4C FE59D32E 5F545012 C566516B 7874FB3D
```

```
AED51403 31F29528 B30FC8B8 A9371C28 18017B09 53A84FFC
9FBFF84B 64BF0238 AA7E2AF2 ECADC15A 1C06DADC F1F2E7B1
240A5E64 5A6469C9 B002215D 9A91C2A4 ED2FB547 A942D777
```

F.3.2.3 Signature key and verification key

X = 7C28569A 94B46FA7 45C8D306 AD7DC189 96CE046E EBE04383

```
8391C232 078DB05A
```

Y = 2574E10E 806F1C42 58F7CF8F A4A6CF2B EB177DBE 60E4EC17

```
DF21DCDB A72073F6 5565506D A3DF98D5 A6C8EEE6 1B6B5D88
B98C47C2 B2F6FC6F 504FA4FB C7F411E2 3EAA3B18 7A353DAE
D41533A9 558AB932 0A154CAE CC544E43 0008889A 2C899373
EC75A24C FF26247C F297D293 747ECC05 B3483647 A87BCBB8
D4500092 09F5E449 A00A659B 637CE139 CF6487AC A70F9C00
CB670C7F 3B95BFD7 CF236A0A 6F3C93BE 8D9CF591 C9D30686
9415B1AA 97264B90 4167850A 4794C780 BE4527DF FEB67BE6
E66786C5 CCE0378C CB49920D 855558F4 DAC4C42F 92DD229B
483B2257 DB0CE35D C737F980 1A261A02 BDF718C2 FD4D69C5
2E0D9712 B42C4897 BAE7C684 D3D35BC5 726CE899 2696B044
D722AFBA 78EFA858 C4D10F19 72112CE8 FFD39792 49BF14E4
9D8E0D9A CB1B0A9C A90D0551 1803845D 7C670BCF 1B066497
A7743B08 A219E764 EA0A3A2A 617661C1 6A372FE0 58B547A2
8B626ECF 442222E1 8EEF487C C101DBFB 715BC33A B85928EC
F0BD4DEA 30F250A6 A5C86178 83EA0F87 3E7A4651 98C4644B
```

F.3.2.4 Per message data

M = ASCII form of "This is a test message for KCDSA usage!" =

```
54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6D
65 73 73 61 67 65 20 66 6F 72 20 4B 43 44 53 41
20 75 73 61 67 65 21
```

K = 83F3008F CEBAE57E C7A64A3A F7EE6EE1 9CC197A6 D5EBA3A5

```
B3EF79B2 F8F3DD53
```

Y' = EA0A3A2A 617661C1 6A372FE0 58B547A2 8B626ECF 442222E1

8EEF487C C101DBFB 715BC33A B85928EC F0BD4DEA 30F250A6
A5C86178 83EA0F87 3E7A4651 98C4644B

$h(Y'\|M)$ = 4D4F2A9B 83446B62 F571A669 FACB2D30 7ADE18DE 1A3FFB87
649ABA4E 606A0751

V = 1935B399 849AB60F 0AE62FAD 82B281E9 1A098A8F 51E6E7D6
BA581801 F02604A0

F.3.2.5 Signature

R = 547A9902 07DEDD6D FF9789C4 7879ACD9 60D79251 4BD91C51
DEC2A24F 904C03F1

S = 1668797B 26641E72 94AA68D3 8562EAE3 CAA842D0 F446949C
4268AE3D 0392434F

F.3.2.6 Verification

R' = 547A9902 07DEDD6D FF9789C4 7879ACD9 60D79251 4BD91C51
DEC2A24F 904C03F1

F.3.3 Example 3: 2048-bit Prime P, 224-bit Prime Q, SHA-256

F.3.3.1 General

This example uses SHA-256 as the hash-function h. The hash-code is simply the value of SHA-256.

F.3.3.2 Parameters

l = 200 (i.e. 512 in decimal)

α = 2048

β = 224

P = 8DA8C1B5 C95D11BE 46661DF5 8C9F803E B729B800 DD92751B
3A4F10C6 A5448E9F 3BC0E916 F042E399 B34AF9BE E582CCFC
3FF5000C FF235694 94351CFE A5529EA3 47DCF43F 302F5894
380709EA 2E1C416B 51A5CDFC 7593B18B 7E3788D5 1B9CC9AE
828B4F8F B06E0E90 57F7FA0F 93BB0397 031FE7D5 0A6828DA
0C1160A0 E66D4E5D 2A18AD17 A811E70B 14F4F431 1A028260
3233444F 98763C5A 1E829C76 4CF36ADB 56980BD4 C54BBE29
7E790228 4292D75C A3600FF4 59310B09 291CBEFB C721528A

```
        13403B8B 93B711C3 03A2182B 6E6397E0 83380BF2 886AF3B9
        AFCC9F50 55D8B713 6C0EBD08 C5CF0B38 888CD115 72787F6D
        F384C97C 91B58C31 DEE5655E CBF3FA53
    Q = 864F1884 1EC103CD FD1BE7FE E54650F2 2A3BB997 537F32CC
        79A51F53
    G = 0E9BE1F8 7A414D16 7A9A5A96 8B079E4A D385A357 3EDB21AA
        67A6F61C 0D00C14A 7A225044 B6E9EB03 68C1EB57 B24B45CD
        854FD93C 1B2DFB0A 3EA302D2 367E4EC7 2F6E7EE8 EA7F8002
        F7704E99 0B954F25 BADA8DA6 2BAEB6F0 6953C0C8 5104AD03
        F36618F7 6C62F4EC F3480183 69850A56 17C999DB E68BA17D
        5BC72556 74EF4839 22C6A3F9 9D3C3C6F 358896C4 E63C605E
        E7DB16FC BD9BE354 E281F7FE 7813D054 27ED1912 B5C7653A
        167B9434 9147EEAF 85CC9CE2 E81661F3 21512D5D 2C0580B0
        3D1704EE F2317F45 185C8258 387E7EC9 79C04707 EF546241
        2784AFE4 1A7B45C8 3B9CBE48 F9127CB4 400BE9E9 6AC5DE17
        F2C9DEA3 5E3734E7 9B64673F 85681C4E
```

F.3.3.3 Signature key and verification key

```
    X = 2F1991C1 AF401872 8A5A431B 9B5459DF B16F6D25 6797FE57
        0EC6BC65
    Y = 04EDE5C6 7EA29297 A8CACB6B DE6F4666 AEA27D10 3DD1E9E9
        582F76A2 F22B8B1B 32230BC5 8F06B768 F8102B49 FA1CAE5E
        18921494 7F6239B6 C6CE7C9B C2D230E8 9A40BEE2 C33A8861
        FD4F7D35 B788FE95 B2D5885D 8C8FAEA8 1C90BE4C EE2784E3
        3577A71D 3B7F085D 71E9A1D4 7815C73F A087ACAA B9FCB565
        5AC9570E 6852BE7C 9C0AECEA 8BD9AA75 A44FC314 7F733E90
        6ADB0FD7 6D613561 B1DB364B BDC9AFD3 CE8F5F17 E3E71203
        4A999350 8059FA52 441FA90D DFE9A0F2 A0B9192F E2220C08
        1BD0C0F0 E07CB5F1 EE4FF405 23591F17 8A4FC7CB 5065F6A3
        8216E9A0 99C205B2 9B8746D8 65E1AF6D 903E5A13 8004910B
        70EB5B84 EED9760E A60578BF 08852898
```

F.3.3.4 Per message data

M = ASCII form of "This is a test message for KCDSA usage!" =

```
54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6D
65 73 73 61 67 65 20 66 6F 72 20 4B 43 44 53 41
20 75 73 61 67 65 21
```

K =
```
49561994 FD2BAD5E 410CA1C1 5C3FD3F1 2E70263F 2820AD5C
566DED80
```

Y' =
```
1BD0C0F0 E07CB5F1 EE4FF405 23591F17 8A4FC7CB 5065F6A3
8216E9A0 99C205B2 9B8746D8 65E1AF6D 903E5A13 8004910B
70EB5B84 EED9760E A60578BF 08852898
```

$I2BS(\beta, BS2I(\gamma, h(Y' || M_2)) \ mod \ 2^{\beta})$ =

```
894315A9 A68EF862 7D015AAE 4EBB41B3 FED88AAA 9614CC67
09DE2B47
```

V =
```
489142E8 F4A1ACE1 4F693AFD 632A6FB7 5A8392AD E61F8A26
3F665A1B
```

F.3.3.5 Signature

R =
```
C1D25741 522F5483 32686053 2D912E04 A45B1807 700B4641
36B8715C
```

S =
```
0AFAEA63 92942822 86B0F9E1 9EC1BC13 BFA29B54 5747F262
0DA3AFC1
```

F.3.3.6 Verification

R' =
```
C1D25741 522F5483 32686053 2D912E04 A45B1807 700B4641
36B8715C
```

F.4 Pointcheval-Vaudenay mechanism

F.4.1 Example 1: 2048-bit Prime P, SHA-224

F.4.1.1 General

This example uses SHA-224 as the hash-function h. The hash-code is simply the value of SHA-224.

F.4.1.2 Parameters

```
L = 200 (i.e. 512 in decimal)

F = 2

P = EEADFD4F 9CC4EFF7 F5B3F5D9 047399E1 00E6D01D 6DE30E40
    39B25F76 083743F1 826919FE D5C750F5 BC6AD9C1 C0B9E229
    695F306C 1AE9F631 7EC4AC94 74FD88F9 714798E6 2FD2D865
    A2EE7BB4 E6C38A8F 2A4C1C30 1FEC7149 1356AD5A 87449AFF
    F1AE412B 128ECF26 DB9DBD74 C91CBDCA 4ED4CE4F 43AC2770
    E7B38648 6190B6A4 0211E61F F5F767C2 31F9FDEF FE999F47
    BA2C2111 821C3EE5 BDD1BBC2 E4C89E5A 5C54DC16 0167A8D6
    B0235223 44B45E72 AB2A54A7 A3787C59 6A38AA76 3589852C
    50D67BFB 469278F0 7E6F5C03 2785FA22 2E635460 65E5E1D5
    95BE7E94 F01C274C B3291116 40544989 38824CC9 55D802CE
    76C0FD6E 33537111 78C33059 6F25CF05

Q = C3E87841 9B4463CE 4FBA73B6 022498A0 EA51A6B8 050E79BB
    A5FEF99D

G = BC376597 C582A659 072A5F1A 2839A1DF 594B0AE2 1262A447
    A8A0F6CB 60596837 661C1734 DFC2BB0B 9F756AA3 44AD8ADD
    A15193A9 6CA727E3 A9D8F32E B20E9760 178862DF C154A308
    2EAD2E4A 2F7C760A 61E86750 D8B3CFDC 3D00A46A E787A3DE
    5F657ADA 64CD0BFF A3B2B228 88299DE3 100F58B7 CD593D82
    E5B31614 997FF1CA 6E87790A 3C8B551B 82606A1C BC80ACD2
    37683EE2 427BA87C 06FFC4EC 17504261 E99DE627 8F9EF77B
    A6E49A82 9C9F27E9 87812CD2 8A15EEAA 36391E67 3042E8A2
    9C76C8A4 DABFDB3D 48D351F5 8B870F9F 609AE941 FFA9AA2D
    886045F9 5EE9B836 9E1C9545 BC18D60C AE7F56AE 8A24B387
    395F0CFC 99E36A65 AFB8C269 522352A0
```

F.4.1.3 Signature key and verification key

```
X = 26534136 393884D5 885F924C 188C83AC 860A2560 17B323B4
    118E1B1A

Y = D88E9870 78469EB9 B6296190 64BF87C8 1F5BFFD3 88C03AB8
```

```
C6C2048E 5B8D087C 8C1D7A5F E2D3394E F567CA43 8AC89B5A
27F26DF2 CCBFDB4A 4351E2BA 42866D82 72DB1BB2 422157D5
3659BB68 FE50F61C 31573684 E93B19C5 F3F832C5 01B6CD9F
3F0E40FA 9198D358 F2BD39F4 DDC4CC5E BD670CB1 677F4F8F
D0867B95 D9AB1A8E 2A8377A9 8189FEF4 0B24150B 2F7EAD42
BBD61A86 C6150431 87614940 435E55DB 55CA0F37 C6F06142
0577EB51 196E0021 431C94DB D746800A F1B4F23E 86F40B57
AB964428 872EE96D A99E3576 0613D279 2FC3E8A8 0CDD1877
117B9ED5 401511ED 43AC10BF 1E1F7450 01AB51CF 842EF938
386947FE BD4CBE9B 1FC3D785 C937D5CE
```

F.4.1.4 Per message data

K = 717548D2 2C7FAF7E BE1BD5DC AD2C2E47 5DA789D5 68C5E081 236214D0

K^{-1} = 787CEDE9 BD149C67 45C83654 E1F9B472 C736D584 ECD4ACE4 4943FD61

M = ASCII form of "abc" = 61 62 63

F.4.1.5 Signature

R = AF246E82 74F24075 343014D5 FC648CE0 09771BD0 48DF1438 EB0D97E7

$R||M$ = AF246E82 74F24075 343014D5 FC648CE0 09771BD0 48DF1438 EB0D97E7 616263

$h(R||M)$ = 5986F5F E1D8BF68 2BD8FE31 25065C4C D9A77CDD 73EE5E85 01D29EF2

S = 370F4FD9 3E761FD9 B97DC37D 4DAD75E2 4EF3A6FD 3D5AC9F1 867A7E33

F.4.1.6 Verification

Π' = 18A358C2 2635ABE3 85E18D55 AF94B75B 36EC2FAE 8E87EA24

```
C0BBC507 A8790BA9 0CD0A93D C92E4736 E5A050BD DA2BF62E
AA042C69 07D39346 D909753C 4C91024F DC12D5AA 5A98FE7E
EE3A90AD 64DB83E9 7C91EAC5 CDC41AA0 D383B3BA 094B0A58
```

```
       C64BF470 0FC81437 BAD1A15D 3FD61B53 25E2991E D841D159
       EEF740DE C7984657 D4B0BE24 9DB622E1 0936BE55 2AE4005A
       59DFA542 A688F0DE 3C5D67EB C5A37ED4 C5A3D0F1 CAF1B8FC
       24E7E658 368FF8C1 E10A3F80 BE42B30C 6255B2AE 15F0251C
       CF1CD9F5 41ECCCF7 7E6A6147 B79FE310 09C8172D BD0CD110
       52816FA7 04FADDA3 5C73E38F 9C6BD8BA 131A2073 191D9D64
       94A3E764 4ED385FF E01D85BD A015FB6C
```

R' =
```
       AF246E82 74F24075 343014D5 FC648CE0 09771BD0 48DF1438
       EB0D97E7
```

F.5 SDSA mechanism

F.5.1 Example 1: 2048-bit Prime P, SHA-224

F.5.1.1 General

For this example, the ‘2048-bit MODP group with 224-bit prime order subgroup‘ in RFC-5114 is used as the group. SHA-224 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-224, converted according to Annex B to the appropriate data item.

F.5.1.2 Parameters

α = 2048

β = 224

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

F.5.1.3 Signature key and verification key

```
  X = 602FE736 80BEFCB2 A8B46779 35FF652B 21A3F4DE 46725D07
      D7D371A9
  Y = A7DBB446 FD8C4B82 61BE026D 94FA9847 74B17110 CABC0944
      14AD2013 2EFC8B7D 7C7FF05D D0B902C4 EF736831 61C1F9A3
      9D60E7AB E3BD9FE2 B458A96D F4783408 0AA93CAF 09673967
      F434548D 44B278E0 4C1FA6D5 E0C41990 CEF37940 66015ED4
      748DAD56 429596DD 9259C45C 21B71A5E A4EF099A 06DAC737
      8958A107 B11B3E57 384118E0 19897C48 E734F069 E717E23A
      DD202405 823A2AE7 A08AFA51 09D2CF6A 0FF546FA 38A8735D
      1CE715E0 6AC08EB0 93EB331F EBEC88D6 1DF546E2 DC9E8465
      10B63F6A 5BA73FE3 6995BD17 1B9D7D35 9EEB3D7D B801F382
      1D582280 DF71A27D 5191E4AB 42BE27A3 1180F537 CABAC0D1
      2EBF1698 B1884697 9EDCA15D DC8F86DC
```

F.5.1.4 Per message data

M = ASCII form of "abc" = 616263

```
  K = 7BFA2DD5 6B31BB27 FFC0D1AE 1ABAA90F A0BB9379 08A542A1
      5EFD1E15
  Π = 60FCB613 40799851 B5E2DC3A 3865BC21 29100D38 4B1C9A94
```

```
      6F0C873B 442BEBD8 5904CD09 A4C6A29E 0CD1111E B9E65F82
      85F8A578 A5717098 FA2A601F D9183CDD D5FF1586 AB255E1D
      4DF4A141 DFE717DC 16DA3B0D 438B1EA5 4976523F 1D73351B
      F39B1987 97DA0EC7 E9EE994A 4C0352D8 271D186A 0DEA8AB0
      FD5E7862 17016E91 03C5F139 2C1D3C01 B974BADC 88184905
      065F8DA8 55656BAF B3B1EDBC 4C14A969 2AEA1A71 D85117F4
      08548EF5 A34966B9 0123FC81 72472B44 06D0C2E6 77C3C21D
      D0680C63 0DC69BFF BA67D89F F17CA52A 6B0F164F 5452777D
      B838BDBD EE60E03B AE773475 42435BD9 09D021DD F97602E0
      3FE41463 CC18128B AFA6E661 6F6CA744
  R = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3
      316AD681
```

F.5.1.5 Signature

```
  R = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3
      316AD681
  S = 4C776699 9F9D52E1 52B47F29 335E548F A3B90625 C55FC9A4
      FE2D5F1F
```

F.5.1.6 Verification

```
 Π′ = 60FCB613 40799851 B5E2DC3A 3865BC21 29100D38 4B1C9A94
      6F0C873B 442BEBD8 5904CD09 A4C6A29E 0CD1111E B9E65F82
      85F8A578 A5717098 FA2A601F D9183CDD D5FF1586 AB255E1D
      4DF4A141 DFE717DC 16DA3B0D 438B1EA5 4976523F 1D73351B
      F39B1987 97DA0EC7 E9EE994A 4C0352D8 271D186A 0DEA8AB0
      FD5E7862 17016E91 03C5F139 2C1D3C01 B974BADC 88184905
      065F8DA8 55656BAF B3B1EDBC 4C14A969 2AEA1A71 D85117F4
      08548EF5 A34966B9 0123FC81 72472B44 06D0C2E6 77C3C21D
      D0680C63 0DC69BFF BA67D89F F17CA52A 6B0F164F 5452777D
      B838BDBD EE60E03B AE773475 42435BD9 09D021DD F97602E0
```

3FE41463 CC18128B AFA6E661 6F6CA744

R' = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3

316AD681

F.5.2 Example 2: 2048-bit Prime P, SHA-256

F.5.2.1 General

For this example, the '2048-bit MODP group with 256-bit prime order subgroup' in RFC-5114 is used as the group. SHA-256 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-256, converted according to Annex B to the appropriate data item.

F.5.2.2 Parameters

α = 2048

β = 256

p = 87A8E61D B4B6663C FFBBD19C 65195999 8CEEF608 660DD0F2

5D2CEED4 435E3B00 E00DF8F1 D61957D4 FAF7DF45 61B2AA30

16C3D911 34096FAA 3BF4296D 830E9A7C 209E0C64 97517ABD

5A8A9D30 6BCF67ED 91F9E672 5B4758C0 22E0B1EF 4275BF7B

6C5BFC11 D45F9088 B941F54E B1E59BB8 BC39A0BF 12307F5C

4FDB70C5 81B23F76 B63ACAE1 CAA6B790 2D525267 35488A0E

F13C6D9A 51BFA4AB 3AD83477 96524D8E F6A167B5 A41825D9

67E144E5 14056425 1CCACB83 E6B486F6 B3CA3F79 71506026

C0B857F6 89962856 DED4010A BD0BE621 C3A3960A 54E710C3

75F26375 D7014103 A4B54330 C198AF12 6116D227 6E11715F

693877FA D7EF09CA DB094AE9 1E1A1597

G = 3FB32C9B 73134D0B 2E775066 60EDBD48 4CA7B18F 21EF2054

07F4793A 1A0BA125 10DBC150 77BE463F FF4FED4A AC0BB555

BE3A6C1B 0C6B47B1 BC3773BF 7E8C6F62 901228F8 C28CBB18

A55AE313 41000A65 0196F931 C77A57F2 DDF463E5 E9EC144B

777DE62A AAB8A862 8AC376D2 82D6ED38 64E67982 428EBC83

1D14348F 6F2F9193 B5045AF2 767164E1 DFC967C1 FB3F2E55

A4BD1BFF E83B9C80 D052B985 D182EA0A DB2A3B73 13D3FE14

C8484B1E 052588B9 B7D2BBD2 DF016199 ECD06E15 57CD0915

B3353BBB 64E0EC37 7FD02837 0DF92B52 C7891428 CDC67EB6

184B523D 1DB246C3 2F630784 90F00EF8 D647D148 D4795451

5E2327CF EF98C582 664B4C0F 6CC41659

q = 8CF83642 A709A097 B4479976 40129DA2 99B1A47D 1EB3750B

A308B0FE 64F5FBD3

F.5.2.3 Signature key and verification key

X = 73018895 20D47AA0 55995BA1 D8FCD701 6EA62E09 18892E07

B7DC23AF 69006B88

Y = 57A17258 D4A3F47C 4545AD51 F3109C5D B41B7878 79FCFE53

8DC1DD5D 35CE42FF 3A9F225E DE650212 6408FCB1 3AEA2231

80B149C4 64E176EB F03BA651 0D8206C9 20F6B1E0 9392E6C8

40A05BDB 9D6875AB 3F4817EC 3A65A665 B788ECBB 447188C7

DF2EB4D3 D9424E57 D964398D BE1C6362 659C6BD8 55C1D3E5

1D64796C A598480D FDD9580E 55085345 C15E34D6 A33A2F43

E222407A CE058972 D34952AE 2B705C53 2243BE39 4B222329

6161145E F2927CDB C55BBD56 4AAE8DE4 BA4500A7 FA432FE7

8B0F0689 1E408083 7E761057 BC6CB8AC 18FD4320 7582032A

FB63C624 F32E66B0 5EC31C5D FFB25FA9 2D4D00E2 B0D4F721

E88C417D 2E57797B 8F55A2FF C6EE4DDB

F.5.2.4 Per message data

M = ASCII form of "abc" = 616263

K = 2B73E8FF 3A7C0168 6CA556E0 FABFD74A C8D1FDA4 AD3D503F

23B8EB8A EEC63305

Π = 41979DBA 19606871 A25BBB51 665AD584 9511D125 8C077E93

027D79AC 35EE887C 460C2689 3D7FFC59 0F317B7A 2C01FCB4

3667E373 F8F2D239 0C950BAF 972E6E0B 00A79416 9696CC95

08A895D6 3F8AA7EA 564AA5DE 6104920A 5E9F687F 469BC831

0C02BE30 B8FD4A54 A167E114 01FEE171 EF284811 6146CC69

C0899526 7186C43E 98B5E62E 9CE5C344 646950EF F57B1490

27FE078F CD9C8297 4AFF1DDE 5AF18E4C A3E3787F 66D15D23

292B5F4E 225AAC64 FC904AB3 40A88B76 40F2D436 D2840185

077EACA4 EE75113E 95B26149 F7C6D2CD 554463F9 26E48F09

AD3C99B8 5EA3EC39 E795FEAE C90C8293 FB0D0506 0FE2BF91

5F00E2FB 7C17B2E8 7C462ED0 D49B8E2F

R = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3

D6DB0CE6 E0ED696E

F.5.2.5 Signature

R = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3

D6DB0CE6 E0ED696E

S = 3505AEA2 E039E18F DDC6580A E89E15DF 0103FB45 C1BB763E

DA4EE6F5 F01783CE

F.5.2.6 Verification

Π' = 41979DBA 19606871 A25BBB51 665AD584 9511D125 8C077E93

027D79AC 35EE887C 460C2689 3D7FFC59 0F317B7A 2C01FCB4

3667E373 F8F2D239 0C950BAF 972E6E0B 00A79416 9696CC95

08A895D6 3F8AA7EA 564AA5DE 6104920A 5E9F687F 469BC831

0C02BE30 B8FD4A54 A167E114 01FEE171 EF284811 6146CC69

C0899526 7186C43E 98B5E62E 9CE5C344 646950EF F57B1490

27FE078F CD9C8297 4AFF1DDE 5AF18E4C A3E3787F 66D15D23

292B5F4E 225AAC64 FC904AB3 40A88B76 40F2D436 D2840185

077EACA4 EE75113E 95B26149 F7C6D2CD 554463F9 26E48F09

AD3C99B8 5EA3EC39 E795FEAE C90C8293 FB0D0506 0FE2BF91

5F00E2FB 7C17B2E8 7C462ED0 D49B8E2F

R' = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3

D6DB0CE6 E0ED696E

F.6 EC-DSA mechanism

F.6.1 General

For the following examples, SHA-1 is used exclusively for the hash-function, so that the hash-code is simply the value of SHA-1, converted according to Annex B to the appropriate data item.

From a security viewpoint, it is important to avoid cryptographically weak curves (e.g. it should be ensured that a particular curve is not vulnerable to attacks on special instances of the elliptic curve discrete logarithm problem).

F.6.2 Example 1: Field F_{2^m}, m =191, SHA-1

F.6.2.1 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{191} + x^9 + 1$.

The elliptic curve is: $Y^2 + X Y = X^3 + a X^2 + b$ over F_{2^m}.

a = 2866537B 67675263 6A68F565 54E12640 276B649E F7526267

b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC

$G = (G_X, G_Y)$

G_X = 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D

G_Y = 765BE734 33B3F95E 332932E7 0EA245CA 2418EA0E F98018FB

Q = 40000000 00000000 00000000 04A20E90 C39067C8 93BBB9A5

F.6.2.2 Signature key and verification key

X = 340562E1 DDA332F9 D2AEC168 249B5696 EE39D0ED 4D03760F

$Y = (Y_X, Y_Y)$

Y_X = 5DE37E75 6BD55D72 E3768CB3 96FFEB96 2614dEA4 CE28A2E7

Y_Y = 55C0E0E0 2F5FB132 CAF416EF 85B229BB B8E13520 03125BA1

F.6.2.3 Per message data

M = ASCII form of "abc" = 61 62 63

K = 3EEACE72 B4919D99 1738D521 879F787C B590AFF8 189D2B69

$\Pi = (\Pi_X, \Pi_Y)$

Π_X = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

Π_Y = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA

$h(M)$ = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

F.6.2.4 Signature

R = 038E5A11 FB55E4C6 5471DCD4 998452B1 E02D8AF7 099BB930

S = 0C9A08C3 4468C244 B4E5D6B2 1B3C6836 28074160 20328B6E

F.6.2.5 Verification

$\Pi' = (\Pi'_X, \Pi'_Y)$

Π'_X = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

Π'_Y = 2AD603A0 5BD1D177 649F9167 E6F475b7 E2FF590C 85AF15DA

R' = 038E5A11 FB55E4C6 5471DCD4 998452B1 E02D8AF7 099BB930

F.6.3 Example 2: Field F_P, 192-bit Prime P, SHA-1

F.6.3.1 Parameters

The field is F_P where P is in hexadecimal

P = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF

The elliptic curve is: $Y^2 = X^3 + a\,X + b$ over F_P.

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC

b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

$G = (G_X, G_Y)$

G_X = 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

G_Y = 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

Q = FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831

F.6.3.2 Signature key and verification key

X = 1A8D598F C15BF0FD 89030B5C B1111AEB 92AE8BAF 5EA475FB

$Y_X = (Y_X, Y_Y)$

Y_X = 62B12D60 690CDCF3 30BABAB6 E69763B4 71F994DD 702D16A5

Y_Y = 63BF5EC0 8069705F FFF65E5C A5C0D697 16DFCB34 74373902

F.6.3.3 Per message data

M = ASCII form of "abc" = 616263

$h(M)$ = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

K = FA6DE297 46BBEB7F 8BB1E761 F85F7DFB 2983169D 82FA2F4E

$\Pi = (\Pi_X, \Pi_Y)$.

Π_X = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

Π_Y = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835.

F.6.3.4 Signature

```
R = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

S = E9ECC781 06DEF82B F1070CF1 D4D804C3 CB390046 951DF686
```

F.6.3.5 Verification

$\Pi' = (\Pi'_X, \Pi'_Y)$.

Π'_X = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

Π'_Y = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835

R' = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

F.6.4 Example 3: Field F_{2^m}, m =283, SHA-256

F.6.4.1 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{283} + x^{12} + x^7 + x^5 + 1$.

The elliptic curve is:$Y^2 + XY = X^3 + aX^2 + b$ over F_{2^m}.

a = 1

b = 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2
A581485A F6263E31 3B79A2F5

$G = (G_X, G_Y)$

G_X = 05F93925 8DB7DD90 E1934F8C 70B0DFEC 2EED25B8 557EAC9C
80E2E198 F8CDBECD 86B12053

G_Y = 03676854 FE24141C B98FE6D4 B20D02B4 516FF702 350EDDB0
826779C8 13F0DF45 BE8112F4

Q = 03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFEF90 399660FC
938A9016 5B042A7C EFADB307

F.6.4.2 Signature key and verification key

X = 010652D3 7B0A9DB6 4D4033AC 6549CD1D F37E1EED E2612C23
63257C6A FF6C8CB5 DCB63648

$Y_X = (Y_X, Y_Y)$

Y_X = 0390858E 9327A714 C74AF0C3 ADEDF4E6 C75CAFDC C46507A4
9E415B13 8A094B6F 43E882AC

Y_Y = 00D4A65D 973CD150 A5221BED F872A4BA 207FF442 7DFFFD48
27C5BF16 9E719162 504D0631

F.6.4.3 Per message data

M = ASCII form of "Example of ECDSA with B-283"

$h(M)$ = F0BF4AEF 3F694EBD DE0A7944 5C897ADB 2430B918 77C772DA
9B7362CB 03AEA87F

K = 0100EC32 1393E6DD 6C4D47BE 5AE189E5 E3540857 9D086217
8F94CCBB A3C4049A 4D88E297

$\Pi = (\Pi_X, \Pi_Y)$.

Π_X = 077CB284 AC41E72E DA2A93EB 8D6DFF58 620F6C69 D528DFE9
0D909AA5 CABC03A3 4E5D5A76

F.6.4.4 Signature

R = 037CB284 AC41E72E DA2A93EB 8D6DFF58 620F7CD9 9B927EEC
7A060A8F 6FB7D926 5EAFA76F

S = 00A37AC1 0AEBFC22 FC6E6EE2 2E8F235E 3EEB0555 A0F0F9DA
92D9FFA7 34AD7679 56D27F23

F.6.4.5 Verification

$\Pi' = (\Pi'_X, \Pi'_Y)$

Π'_X = 077CB284 AC41E72E DA2A93EB 8D6DFF58 620F6C69 D528DFE9
0D909AA5 CABC03A3 4E5D5A76

R' = 037CB284 AC41E72E DA2A93EB 8D6DFF58 620F7CD9 9B927EEC
7A060A8F 6FB7D926 5EAFA76F

F.6.5 Example 4: Field F_p, 256-bit Prime P, SHA-256

F.6.5.1 Parameters

The field is F_P where P is in hexadecimal

P = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF
FFFFFFFF

The elliptic curve is: $Y^2 = X^3 + aX + b$ over F_P.

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC

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

G = (G_X, G_Y)

G_X = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0
F4A13945 D898C296

G_Y = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE
CBB64068 37BF51F5

Q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84
F3B9CAC2 FC632551

F.6.5.2 Signature key and verification key

X = C477F9F6 5C22CCE2 0657FAA5 B2D1D812 2336F851 A508A1ED
04E479C3 4985BF96

Y = (Y_X, Y_Y)

Y_X = B7E08AFD FE94BAD3 F1DC8C73 4798BA1C 62B3A0AD 1E9EA2A3
8201CD08 89BC7A19

Y_Y = 3603F747 959DBF7A 4BB226E4 19287290 63ADC7AE 43529E61
B563BBC6 06CC5E09

F.6.5.3 Per message data

M = ASCII form of "Example of ECDSA with P-256"

$h(M)$ = A41A41A1 2A799548 211C410C 65D8133A FDE34D28 BDD542E4
B680CF28 99C8A8C4

K = 7A1A7E52 797FC8CA AA435D2A 4DACE391 58504BF2 04FBE19F
14DBB427 FAEE50AE

Π = (Π_X, Π_Y).

Π_X = 2B42F576 D07F4165 FF65D1F3 B1500F81 E44C316F 1F0B3EF5
7325B69A CA46104F

F.6.5.4 Signature

R = 2B42F576 D07F4165 FF65D1F3 B1500F81 E44C316F 1F0B3EF5
7325B69A CA46104F

S = DC42C212 2D6392CD 3E3A993A 89502A81 98C1886F E69D262C
4B329BDB 6B63FAF1

F.6.5.5 Verification

Π' = (Π'_X, Π'_Y)

Π'_X = 2B42F576 D07F4165 FF65D1F3 B1500F81 E44C316F 1F0B3EF5
7325B69A CA46104F

R' = 2B42F576 D07F4165 FF65D1F3 B1500F81 E44C316F 1F0B3EF5
7325B69A CA46104F

F.7 EC-KCDSA mechanism

F.7.1 Example 1: Field $\mathbf{F}_P$, 224-bit Prime P, SHA-224

F.7.1.1 General

This example uses SHA-224 as the hash-function h. The hash-code is simply the value of SHA-224.

F.7.1.2 Parameters

The field is $\mathbf{F}_P$ where P is

P = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000
00000001

The elliptic curve is: $Y^2 = X^3 + a\,X + b$ over F_P.

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF
FFFFFFFE

b = B4050A85 0C04B3AB F5413256 5044B0B7 D7BFD8BA 270B3943
2355FFB4

G = (G_X, G_Y)

G_X = B70E0CBD 6BB4BF7F 321390B9 4A03C1D3 56C21122 343280D6
115C1D21

G_Y = BD376388 B5F723FB 4C22DFE6 CD4375A0 5A074764 44D58199
85007E34

Q = FFFFFFFF FFFFFFFF FFFFFFFF FFFF16A2 E0B8F03E 13DD2945
5C5C2A3D

F.7.1.3 Signature key and verification key

X = 562A6F64 E162FFCB 51CD4707 774AE366 81B6CEF2 05FE5D43 912956A2

$Y = (Y_X, Y_Y)$

Y_X = B574169E 4FCEF1AF 3429D8BB 5481FF7D FA978690 492E1098 B80A5579

Y_Y = 1576819B D9F0B685 19EE844A FE88CCFB 2AD574A5 6472D954 1461AE7E

F.7.1.4 Per message data

M = ASCII form of “This is a sample message for EC-KCDSA implementation validation.” =

```
54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D
65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53
41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76
61 6C 69 64 61 74 69 6F 6E 2E
```

K = `76A0AFC1 8646D1B6 20A079FB 223865A7 BCB447F3 C03A35D8`
`78EA4CDA`

Π = $G^k = (\Pi_X, \Pi_Y)$.

Π_X = `F887C158 65203FF7 2C69C113 A457DF64 4F627801 DFF99D1B`
`CCB49C2D`

Π_Y = `ADE18B5B B7118745 017631E5 E54B36C0 332D70B3 CAA8FB10`
`728B66E0`

Y' = `B574169E 4FCEF1AF 3429D8BB 5481FF7D FA978690 492E1098`
`B80A5579 1576819B D9F0B685 19EE844A FE88CCFB 2AD574A5`
`6472D954 1461AE7E 00000000 00000000`

$h(Y' || M)$ = `8C5CB967 71166477 FF84D281 DB766201 2F842138 8AA6FC05`
`282E2E03`

F.7.1.5 Signature

R = `EEA58C91 E0CDCEB5 799B00D2 412D928F DD23122A 1C2BDF43`
`C2F8DAFA`

S = `AEBAB53C 7A44A8B2 2F35FDB9 DE265F23 B89F65A6 9A8B7BD4`
`061911A6`

F.7.1.6 Verification

R' = `EEA58C91 E0CDCEB5 799B00D2 412D928F DD23122A 1C2BDF43`
`C2F8DAFA`

F.7.2 Example 2: Field F_P, 256-bit Prime P, SHA-256

F.7.2.1 General

This example uses SHA-256 as the hash-function h. The hash-code is simply the value of SHA-256.

F.7.2.2 Parameters

The field is $\mathbf{F}_P$ where P is

P = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFF

The elliptic curve is: $Y^2 = X^3 + a\,X + b$ over F_P.

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC

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

$G = (G_X, G_Y)$

G_X = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0
F4A13945 D898C296

G_Y = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE
CBB64068 37BF51F5

Q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84
F3B9CAC2 FC632551

F.7.2.3 Signature key and verification key

X = 9051A275 AA4D9843 9EDDED13 FA1C6CBB CCE775D8 CC9433DE
E69C5984 8B3594DF

$Y = (Y_X, Y_Y)$

Y_X = 148EDDD3 734FD5F1 5987579F 516089A8 C9FEF4AB 76B59D7B
8A01CDC5 6C4EDFDF

Y_Y = A4E2E42C B4372A6F 2F3F71A1 49481549 F68D2963 539C853E
46B94696 569E8D61

F.7.2.4 Per message data

M = ASCII form of "This is a sample message for EC-KCDSA implementation validation." =

```
54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D
65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53
41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76
61 6C 69 64 61 74 69 6F 6E 2E
```

K = 71B88F39 8916DA9C 90F555F1 B5732B7D C636B49C 638150BA
C11BF05C FE16596A

$\Pi = G^k = (\Pi_X, \Pi_Y)$.

Π_X = EC3847B0 CA52038A 823D0230 14546B41 4946EF0A 6EE09228
38948459 5F30E26C

Π_Y = 0640451D 36932442 4ABC681D 65653986 6AD9C494 D26FAC14
69FC2A08 D945F130

Y' = 148EDDD3 734FD5F1 5987579F 516089A8 C9FEF4AB 76B59D7B
8A01CDC5 6C4EDFDF A4E2E42C B4372A6F 2F3F71A1 49481549
F68D2963 539C853E 46B94696 569E8D61

$h(Y' \| M)$ = 681C8ED8 9E8B0E1B C369AA10 6F6B9813 E6338F0C 54BE577A
87623492 52F9BEDF

F.7.2.5 Signature

R = 0EDDF680 601266EE 1DA83E55 A6D9445F C781DAEB 14C765E7
E5D0CDBA F1F14A68

S = 9B333457 661C7CF7 41BDDBC0 835553DF BB37EE74 F53DB699
E0A17780 C7B6F1D0

F.7.2.6 Verification

R' = 0EDDF680 601266EE 1DA83E55 A6D9445F C781DAEB 14C765E7
E5D0CDBA F1F14A68

F.7.3 Example 3: Field F_{2^m}, m=233, SHA-224

F.7.3.1 General

This example uses SHA-224 as the hash-function h. The hash-code is simply the value of SHA-224.

F.7.3.2 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{233} + x^{74} + 1$.

The elliptic curve is: $Y^2 + X\,Y = X^3 + a\,X^2 + b$ over F_{2^m}.

a = 1

b = 0066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42
81FE115F 7D8F90AD

$G = (G_X, G_Y)$

G_X = 00FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36

F8F8EB73 71FD558B

G_Y = 0100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA

36716F7E 01F81052

Q = 0100 00000000 00000000 00000000 0013E974 E72F8A69

22031D26 03CFE0D7

F.7.3.3 Signature key and verification key

X = 00BF 83825505 3DBF499C BE190DE3 5BC14AFC 1EA142F3

5EE69838 5B48D688

$Y = (Y_X, Y_Y)$

Y_X = 01F4 85A65E59 B336E140 1C8A311F 01C92626 C663E69F

12A627E5 3E8F0675

Y_Y = 01BF 338CE75A DFB07DEB D962E1D8 0C101587 269AC995

1B40422B 12E9DA3E

F.7.3.4 Per message data

M = ASCII form of “This is a sample message for EC-KCDSA implementation validation.” =

54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D

65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53

41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76

61 6C 69 64 61 74 69 6F 6E 2E

K = 00F4 F088192E 8EB1CD8B 4ECB3A53 33746B40 EBF16966

A213B18A 176B2F62

$\Pi = G^k = (\Pi_X, \Pi_Y)$.

Π_X = 00E4 5041E7AA 060B8B1A 02A7ACAC DB4E95EF F61F33C0

BB8D6EC2 F1C68BA1

Π_Y = 0155 B3A1DA61 F81E04D5 80D07E92 93DF3D4C 7FE34686

BD157374 4D8D3F18

Y' = 01F485A6 5E59B336 E1401C8A 311F01C9 2626C663 E69F12A6

27E53E8F 067501BF 338CE75A DFB07DEB D962E1D8 0C101587

```
         269AC995 1B40422B 12E9DA3E 00000000
h(Y'||M) = E74B3C74 72F2E97E C31861CA 1773472E 58828A98 026277CB
         00EF36AC
```

F.7.3.5 Signature

```
R = 82EF9427 4AC70A3D AC231E38 AE0F0D31 8FD8E189 EE40A3E0
    61EC80BF
S = 00A8 CD7F7573 BAC3C4C4 00F65FDC CCD46F58 EBFC54CE
    45571075 FD7704DB
```

F.7.3.6 Verification

```
R' = 82EF9427 4AC70A3D AC231E38 AE0F0D31 8FD8E189 EE40A3E0
     61EC80BF
```

F.7.4 Example 4: Field F_{2^m}, m=233 (Koblitz Curve), SHA-224

F.7.4.1 General

This example uses SHA-224 as the hash-function h. The hash-code is simply the value of SHA-224. This example uses a Koblitz curve as an elliptic curve.

F.7.4.2 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{233} + x^{74} + 1$.

The elliptic curve is: $Y^2 + X\,Y = X^3 + a\,X^2 + b$ over F_{2^m}.

a = 0

b = 1

$G = (G_X, G_Y)$

```
G_X = 0172 32BA853A 7E731AF1 29F22FF4 149563A4 19C26BF5
      0A4C9D6E EFAD6126
G_Y = 01DB 537DECE8 19B7F70F 555A67C4 27A8CD9B F18AEB9B
      56E0C110 56FAE6A3
Q = 80 00000000 00000000 00000000 00069D5B B915BCD4
    6EFB1AD5 F173ABDF
```

F.7.4.3 Signature key and verification key

X = 0073 6439374F 72B1C723 AE611CB3 DFBCA0A8 E2C5096B
DB9C2D37 21167B49

Y = (Y_X, Y_Y)

Y_X = 01E9 1DEFBD41 AE655105 E046E03E C13E3860 0E9A2C9
920B8E75 53721605

Y_Y = 0112 9C2706D1 9D134891 C7BAD84A 5600C2AF F86068C4
7497F5BD 498D0B76

F.7.4.4 Per message data

M = ASCII form of “This is a sample message for EC-KCDSA implementation validation.” =
54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D
65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53
41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76
61 6C 69 64 61 74 69 6F 6E 2E

K = 0061 7AA0B7A8 197A2B81 01500BFE 55D5322A 7149E275
F91ADBC7 E30128E4

Π = $G^k = (\Pi_X, \Pi_Y)$.

Π_X = 01BB 9CDB150A 2E5669ED C491320C 3F84E28A 7D6631BC
51127677 A2CF2FEF

Π_Y = 00DA E917793C 12DE86AA 6727C396 A3131B69 33344EDD
B621DD29 BC09B648

Y' = 01E91DEF BD41AE65 5105E046 E03EC13E 38600E9A 2C9A920B
8E755372 16050112 9C2706D1 9D134891 C7BAD84A 5600C2AF
F86068C4 7497F5BD 498D0B76 00000000

$h(Y' || M)$ = FC712972 727661DE B546E86A B6937DB7 D9E61A36 DF5CEA86
044BFF25

F.7.4.5 Signature

R = B164A12F 615CC661 C10B78CB 6E01C9DE 46337C50 C036FAC5 51178752

S = 004A 2109081E B3ADF95C 19FFAE89 5D303B83 147B27C6 EFAE8536 2BFAB89A

F.7.4.6 Verification

R' = B164A12F 615CC661 C10B78CB 6E01C9DE 46337C50 C036FAC5 51178752

F.7.5 Example 5: Field F_{2^m}, m=283, SHA-256

F.7.5.1 General

This example uses SHA-256 as the hash-function h. The hash-code is simply the value of SHA-256.

F.7.5.2 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{283} + x^{12} + x^7 + x^5 + 1$.

The elliptic curve is: $Y^2 + X\,Y = X^3 + \mathrm{a}\,X^2 + b$ over F_{2^m}.

a = 1

b = 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2 A581485A F6263E31 3B79A2F5

G = (G_X, G_Y)

G_X = 05F93925 8DB7DD90 E1934F8C 70B0DFEC 2EED25B8 557EAC9C 80E2E198 F8CDBECD 86B12053

G_Y = 03676854 FE24141C B98FE6D4 B20D02B4 516FF702 350EDDB0 826779C8 13F0DF45 BE8112F4

Q = 03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFEF90 399660FC 938A9016 5B042A7C EFADB307

F.7.5.3 Signature key and verification key

X = 00D64BEC 51F1ADA0 5BBD4F2B 53405B0C E8A1B99C D8DB6309 76A47F76 F08F205E EFC3FBD8

Y = (Y_X, Y_Y)

Y_X = 04313C7E 9C4F80D2 6A287B37 FE7FAA96 BE31F116 2E18BDB4

70CF43D4 DB28DE10 8B007E9F

Y_Y = 0342CCF6 F502F9DF EC208170 24326C26 E867E1FB EC6634CB

17023CA0 222D6112 E0BFA106

F.7.5.4 Per message data

M = ASCII form of "This is a sample message for EC-KCDSA implementation validation." =

54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D

65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53

41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76

61 6C 69 64 61 74 69 6F 6E 2E

K = 00D18E44 CB7F75F8 01277FA5 CF31A268 8CC2F322 2FA9F26E

E8598126 AFEEE4E3 8DD0E08E

Π = $G^k = (\Pi_X, \Pi_Y)$.

Π_X = 01EBE1E7 8BAF9EF6 833189B3 ACC5B3DC 85788292 E0006D90

F8A4E2AE C3027F28 BE47FACA

Π_Y = 0721C5B3 4A038EE1 1DEE2FAE DA84FD46 CBCF37BA 676677BB

AB731AE8 8C52833B AB776F45

Y' = 04313C7E 9C4F80D2 6A287B37 FE7FAA96 BE31F116 2E18BDB4

70CF43D4 DB28DE10 8B007E9F 0342CCF6 F502F9DF EC208170

24326C26 E867E1FB EC6634CB 17023CA0

$h(Y' || M)$ = 148DF2CD 1A4E5437 69F5F0B4 FE07A87A D630C512 A3978248

5B8B8A1A EA50D662

F.7.5.5 Signature

R = 4A23BA73 B29A9010 ACD1E231 3B9A252C E209C7BF 3643926F

A7BF8C87 A8C76D40

S = 03AA4FFF F1F4C3EE BF9C8798 2E717572 71CB7662 BA03463B

8B5F97B0 5C7F7C2C 88A31799

F.7.5.6 Verification

R' = 4A23BA73 B29A9010 ACD1E231 3B9A252C E209C7BF 3643926F

A7BF8C87 A8C76D40

F.7.6 Example 6: Field F_{2^m}, m=283 (Koblitz Curve), SHA-256

F.7.6.1 General

This example uses SHA-256 as the hash-function h. The hash-code is simply the value of SHA-256. This example uses a Koblitz curve as an elliptic curve.

F.7.6.2 Parameters

The field F_{2^m} is represented as polynomials modulo the irreducible polynomial $x^{283} + x^{12} + x^7 + x^5 + 1$.

The elliptic curve is: $Y^2 + X\,Y = X^3 + \mathrm{a}\,X^2 + b$ over F_{2^m}.

a = 0

b = 1

G = (G_X, G_Y)

G_X = 0503213F 78CA4488 3F1A3B81 62F188E5 53CD265F 23C1567A
16876913 B0C2AC24 58492836

G_Y = 01CCDA38 0F1C9E31 8D90F95D 07E5426F E87E45C0 E8184698
E4596236 4E341161 77DD2259

Q = 01FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577
265DFF7F 94451E06 1E163C61

F.7.6.3 Signature key and verification key

X = 014930E6 6B51F09F EEBBAFFC 9111C5CF 8AE406C9 35AC9618
F0A613B9 6D97F7DB 8F6EBA74

Y = (Y_X, Y_Y)

Y_X = 078A6ACD D5F779F2 5E8AB413 965E217F E6B1E63D 4717EEF5
0DC8C59D F7B1A095 BC3027AE

Y_Y = 07B6D962 5F2D9DDF 516B5037 E1E7B115 26E12AC4 E65AD498
CD85D65A 9E915D58 6976C00F

F.7.6.4 Per message data

M = ASCII form of "This is a sample message for EC-KCDSA implementation validation." =

```
54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65 20 6D
65 73 73 61 67 65 20 66 6F 72 20 45 43 2D 4B 43 44 53
41 20 69 6D 70 6C 65 6D 65 6E 74 61 74 69 6F 6E 20 76
61 6C 69 64 61 74 69 6F 6E 2E
```

K = 01EA8FB5 72B7B2DA 7149DCD8 78101ECF 3F296400 E13A0D65
C8B6E558 C0237C6D A55268A1

Π = $G^k = (\Pi_X, \Pi_Y)$.

Π_X = 0227BDFF E74468EE 3A327AAC 7078252F F545113A 1DD9A2E0
7A0D238B AE601410 34D91C33

Π_Y = 02F519CC E08F4ACC 46AB4323 45DD0F69 408E1346 5E017832
94DE4128 4E24D02B D6916937

Y' = 078A6ACD D5F779F2 5E8AB413 965E217F E6B1E63D 4717EEF5
0DC8C59D F7B1A095 BC3027AE 07B6D962 5F2D9DDF 516B5037
E1E7B115 26E12AC4 E65AD498 CD85D65A

$h(Y'||M)$ = 23893E3F 87BA26BB B05E0E9B F83A40B0 14EFB95B C87B3AF4
C34902D6 12C8A2B4

F.7.6.5 Signature

R = E214F3CF 8BBB6E92 F779E6C8 A3424BA8 64734002 5EB49EED
C6016746 81B14AFD

S = 0014CC0B B9245B7A 8BC3C6E0 392AAACE DCED8A61 9D9676E9
73D5244D 7F45E01D B425A93E

F.7.6.6 Verification

R' = E214F3CF 8BBB6E92 F779E6C8 A3424BA8 64734002 5EB49EED
C6016746 81B14AFD

F.8 EC-GDSA mechanism

F.8.1 General

For the following examples Brainpool curves and SHA-2 are used.

F.8.2 Example 1: Field F_P, 192-bit Prime P, SHA-256

F.8.2.1 Parameters

The field is F_P where P is in hexadecimal

P = C302F41D 932A36CD A7A34630 93D18DB7 8FCE476D E1A86297

The elliptic curve is: $Y^2 = X^3 + a X + b$ over F_P.

a = 6A911740 76B1E0E1 9C39C031 FE8685C1 CAE040E5 C69A28EF

b = 469A28EF 7C28CCA3 DC721D04 4F4496BC CA7EF414 6FBF25C9

G = (G_X, G_Y)

G_X = C0A0647E AAB6A487 53B033C5 6CB0F090 0A2F5C48 53375FD6

G_Y = 14B69086 6ABD5BB8 8B5F4828 C1490002 E6773FA2 FA299B8F

Q = C302F41D 932A36CD A7A3462F 9E9E916B 5BE8F102 9AC4ACC1

F.8.2.2 Signature key and verification key

X = 40F95B49 A3B1BF55 311A56DF D3B5061E E1DF6439 84D41E35

Y = (Y_X, Y_Y)

Y_X = 754A8F6C 30D28AE9 A63443C9 7DEC844A 15F797D0 B78FEE03

Y_Y = 63EC81B4 6A9F3833 025037DF E7DCDDE7 AF20C5E7 C6733C35

F.8.2.3 Per message data

M = ASCII form of "brainpoolP192r1" =

62 72 61 69 6E 70 6F 6F 6C 50 31 39 32 72 31

K = 5A966260 96288CC4 69F1704E C05F44D1 EC18BD32 CEB02D5B

Π = (Π_X, Π_Y).

Π_X = A00B0AA2 5DB6AB5C 21B86300 D9BC99F5 6E9DD1B7 F1DC4774

Π_Y = 58C0F50E 2E1F6B01 A50E280E 6DB71637 AE9579BC 1565F369

$h(M)$ = 2AE5880D 61FCA83B 2D4C9281 356B9FD2 F7C21359 BA789FBF
D7068AF2 F9A101EC

H = 2AE5880D 61FCA83B 2D4C9281 356B9FD2 F7C21359 BA789FBF

(H is the truncated SHA-256 hash-code of message M to the same bit length as q.)

F.8.2.4 Signature

R = A00B0AA2 5DB6AB5C 21B86300 D9BC99F5 6E9DD1B7 F1DC4774

S = 634635EF 813247D7 20245C94 09FB20A2 67C560C8 8EB2B07B

F.8.2.5 Verification

R' = A00B0AA2 5DB6AB5C 21B86300 D9BC99F5 6E9DD1B7 F1DC4774

F.8.3 Example 2: Field F_P, 224-bit Prime P, SHA-224

F.8.3.1 Parameters

The field is F_P where P is in hexadecimal

P = D7C134AA 26436686 2A183025 75D1D787 B09F0757 97DA89F5
7EC8C0FF

The elliptic curve is: $Y^2 = X^3 + a\,X + b$ over F_P.

a = 68A5E62C A9CE6C1C 299803A6 C1530B51 4E182AD8 B0042A59
CAD29F43

b = 2580F63C CFE44138 870713B1 A92369E3 3E2135D2 66DBB372
386C400B

G = (G_X, G_Y)

G_X = D9029AD2 C7E5CF43 40823B2A 87DC68C9 E4CE3174 C1E6EFDE
E12C07D

G_Y = 58AA56F7 72C0726F 24C6B89E 4ECDAC24 354B9E99 CAA3F6D3
761402CD

Q = D7C134AA 26436686 2A183025 75D0FB98 D116BC4B 6DDEBCA3
A5A7939F

F.8.3.2 Signature key and verification key

X = 7E75BC2C D573B38A ED0977AD 611763DD 57FB29B2 20883344
B81DF037

Y = (Y_X, Y_Y)

Y_X = 8B29B268 866CEDCD A528F443 CAE7B07B F82BDC59 1A4BA29A
C5E7BA4E

Y_Y = CD7746C1 4DEEA220 EA1BF164 C203C46E 60AF6699 CE6E1448
076B5807

F.8.3.3 Per message data

M = ASCII form of "brainpoolP224r1" =
62 72 61 69 6E 70 6F 6F 6C 50 32 32 34 72 31

K = 5B604F2C 35ED0401 FCA31E88 0CB55C2A 7456E71A 5CBAA8DF
2FC03CA9

$\Pi = (\Pi_X, \Pi_Y)$.

Π_X = 60FBB2B1 5F055CD1 D482ED6D C5069C8F 624A3405 B67D11B3
B65E0234

Π_Y = 2C4359F0 A5A69F5A 29D1C1F3 86C1DCA6 5A47D160 DA1FBFB2
BA5A2FDB

$h(M)$ = AC2AC36A D5DAF131 951BA30B 330722C7 4BCFFF79 0617D1F0
908E06AF

F.8.3.4 Signature

R = 60FBB2B1 5F055CD1 D482ED6D C5069C8F 624A3405 B67D11B3
B65E0234

S = 5A050F05 AF0B106B A3F14696 E6162CA4 6FBABD2C 144419DB
B5BFBDC0

F.8.3.5 Verification

R' = 60FBB2B1 5F055CD1 D482ED6D C5069C8F 624A3405 B67D11B3
B65E0234

F.8.4 Example 3: Field F_P, 256-bit Prime P, SHA-256

F.8.4.1 Parameters

The field is F_P where P is in hexadecimal

P = A9FB57DB A1EEA9BC 3E660A90 9D838D72 6E3BF623 D5262028
2013481D 1F6E5377

The elliptic curve is: $Y^2 = X^3 + a\,X + b$ over F_P.

a = 7D5A0975 FC2C3057 EEF67530 417AFFE7 FB8055C1 26DC5C6C
E94A4B44 F330B5D9

b = 26DC5C6C E94A4B44 F330B5D9 BBD77CBF 95841629 5CF7E1CE
6BCCDC18 FF8C07B6

$G = (G_X, G_Y)$

G_X = 8BD2AEB9 CB7E57CB 2C4B482F FC81B7AF B9DE27E1 E3BD23C2
3A4453BD 9ACE3262

G_Y = 547EF835 C3DAC4FD 97F8461A 14611DC9 C2774513 2DED8E54

5C1D54C7 2F046997

Q = A9FB57DB A1EEA9BC 3E660A90 9D838D71 8C397AA3 B561A6F7

901E0E82 974856A7

F.8.4.2 Signature key and verification key

X = 52B929B4 0297437B 98973A2C 437E8F03 A231EB61 E0CD38FD

AD802F00 D55A13A3

$Y = (Y_X, Y_Y)$

Y_X = 90A53E95 D88397AC 76C7C128 297134D9 4BB52866 AD6474C1

3690A5E1 6848AC0D

Y_Y = 0C31F00F 0B3EAC60 A92A19AD 96E9BA31 3A43E100 D4D68FFF

2B8F1D8C D6790714

F.8.4.3 Per message data

M = ASCII form of "brainpoolP256r1" =

62 72 61 69 6E 70 6F 6F 6C 50 32 35 36 72 31

K = E6421272 DDAB9C20 7B119BDD 10C03861 005752EE ABB3AC97

513041AD E6286D9

$\Pi = (\Pi_X, \Pi_Y)$.

Π_X = 829349E3 B6E1F3E5 15EB9581 BE0F958D CCAAA6B6 8D83BA77

01DD7A08 67E44EA7

Π_Y = 0E927978 E600F907 68B68C9E 572D33EC 2F8D8FC9 D577D743

8FDEE63D 27CE9763

$h(M)$ = DB7A981C 4E37DDE0 AEA27A34 E3179BD6 DF307204 75A7993A

FA93DF1D A7EC9910

F.8.4.4 Signature

R = 829349E3 B6E1F3E5 15EB9581 BE0F958D CCAAA6B6 8D83BA77

01DD7A08 67E44EA7

S = 3DC2F103 296A793E 50DC2266 657470A4 0D2C9EA1 CA797DEA

610042B7 730BBDCE

F.8.4.5 Verification

R' = 829349E3 B6E1F3E5 15EB9581 BE0F958D CCAAA6B6 8D83BA77
01DD7A08 67E44EA7

F.9 EC-RDSA mechanism

F.9.1 Example 1: Field F_P, 256-bit Prime P, SHA-256

F.9.1.1 General

For the following example, SHA-256 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-256, converted according to Annex B to the appropriate data item.

From a security viewpoint, it is important to avoid cryptographically weak curves (e.g. it should be ensured that a particular curve is not vulnerable to attacks on special instances of the elliptic curve discrete logarithm problem).

F.9.1.2 Parameters

The field is F_P where P is in hexadecimal

P = 80000000 00000000 00000000 00000000 00000000 00000000
00000000 00000431

The elliptic curve is: $Y^2 = X^3 + aX + b$ over Fp.

a = 7

b = 5FBFF498 AA938CE7 39B8E022 FBAFEF40 563F6E6A 3472FC2A
514C0CE9 DAE23B7E

$G = (G_X, G_Y)$

G_X = 2

G_Y = 08E2A8A0 E65147D4 BD631603 0E16D19C 85C97F0A 9CA26712
2B96ABBC EA7E8FC8

Q = 80000000 00000000 00000000 00000001 50FE8A18 92976154
C59CFC19 3ACCF5B3

F.9.1.3 Signature key and verification key

X = 7A929ADE 789BB9BE 10ED359D D39A72C1 1B60961F 49397EEE
1D19CE98 91EC3B28

$Y = (Y_X, Y_Y)$

Y_X = 7F2B49E2 70DB6D90 D8595BEC 458B50C5 8585BA1D 4E9B788F

```
        6689DBD8 E56FD80B
```

Y_Y = `26F1B489 D6701DD1 85C8413A 977B3CBB AF64D1C5 93D26627`

```
        DFFB101A 87FF77DA
```

F.9.1.4 Per message data

M = ASCII form of "abc" = 616263

$h(M)$ = `BA7816BF 8F01CFEA 414140DE 5DAE2223 B00361A3 96177A9C`

```
        B410FF61 F20015AD
```

K = `77105C9B 20BCD312 2823C8CF 6FCC7B95 6DE33814 E95B7FE6`

```
        4FED9245 94DCEAB3
```

$\Pi = (\Pi_X, \Pi_Y)$

Π_X = `41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8`

```
        3AD043FD 39DC0493
```

Π_Y = `489C375A 9941A304 9E33B343 61DD2041 72AD98C3 E5916DE2`

```
        7695D22A 61FAE46E
```

F.9.1.5 Signature

R = `41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8`

```
        3AD043FD 39DC0493
```

S = `0A7BA472 2DA5693F 229D175F AB6AFB85 7EC2273B 9F88DA58`

```
        92CED311 7FCF1E36
```

F.9.1.6 Verification

$\Pi' = (\Pi'_X, \Pi'_Y)$

Π'_X = `41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8`

```
        3AD043FD 39DC0493
```

Π'_Y = `489C375A 9941A304 9E33B343 61DD2041 72AD98C3 E5916DE2`

```
        7695D22A 61FAE46E
```

R' = `41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8`

```
        3AD043FD 39DC0493
```