INTERNATIONAL STANDARD

**ISO/IEC 14888-3**

Second edition
2006-11-15

# 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 14888-3:2006(E)

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

Published in Switzerland

 

# Contents

Page

# Foreword

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

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

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

ISO/IEC 14888-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 27, *IT Security techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 14888-3:1998), which has been technically revised. It also incorporates Technical Corrigendum ISO/IEC 14888-3:1998/Cor.1:2001. New mechanisms and object identifiers have been specified.

ISO/IEC 14888 consists of the following parts, under the general title *Information technology — Security techniques — Digital signatures with appendix*:

— *Part 1: General*

— *Part 2: Integer factorization based mechanisms*

— *Part 3: Discrete logarithm based mechanisms*

# 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,
  - 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 – Computational feasibility depends on the specific security requirements and environment.

Digital signature mechanisms are based on asymmetric cryptographic techniques and involve 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.

There are 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 non-randomized (or deterministic); see ISO/IEC 14888-1.
- When, for a given message and signature key, each application of the signature process produces a different signature, the mechanism is said to be randomized.

The eight 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" (which is 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" (see ISO/IEC 9796).

Security of the digital signature mechanisms is based on unsolvable problems, i.e. problems for which, given current knowledge, finding a solution is computationally infeasible, such as the factorization problem and the discrete logarithm problem. ISO/IEC 14888-3 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 – The previous version of ISO/IEC 14888 grouped identity-based mechanisms into Part 2 and certificate-based mechanisms into Part 3, with each of the two parts covering mechanisms based on both the discrete logarithm and the factorisation problems. This revision re-organizes the grouping, so that Part 2 contains integer factoring based mechanisms and Part 3 discrete logarithm based mechanisms.

This part of ISO/IEC 14888 includes eight mechanisms, two of which were in ISO/IEC 14888-3:1998, and three of which are in ISO/IEC 15946-2:2002. The Korean Certificate-based Digital Signature Algorithm (KCDSA) and two mechanisms based on pairing technology are newly added.

The mechanisms specified in this part of ISO/IEC 14888 use a collision resistant hash-function for hashing the entire message (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 document may involve the use of patents.

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

The holder of this patent right 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 may be obtained from:

*ISO/IEC JTC 1/SC 27 Standing Document 8 (SD 8) "Patent Information"*. SD 8 is publicly available at: http://www.ni.din.de/sc27

Further information is available from the identified patent-holders.

| Area | Inventors | Patent | Issue date | Contact address |
|---|---|---|---|---|
| DSA | Kravitz | US 5 231 668 | 1993-07-27 | [no licence required] |

Attention is drawn to the possibility that some of the elements of this document 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.

# 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;
- 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;
- the process of verifying signatures.

The verification of a digital signature requires 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 derive 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 – For certificate-based mechanisms, various PKI standards can be used for key management. For further information, see ISO/IEC 11770-3, ISO/IEC 9594-8 (also known as X.509) and ISO/IEC 15945.

## 2 Normative references

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

ISO/IEC 10118 (all parts), *Information technology — Security techniques — Hash-functions*

ISO/IEC 14888-1:1998, *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 $\boldsymbol{E}$ with the binary operation "$*$" such that

- for all $a, b, c \in \boldsymbol{E}$, $(a * b) * c = a * (b * c)$;
- there exists $e \in \boldsymbol{E}$ with $e * a = a$ for all $a \in \boldsymbol{E}$;
- for all $a \in \boldsymbol{E}$ there exists $b \in \boldsymbol{E}$ with $b * a = e$;
- for all $a, b \in \boldsymbol{E}$, $a * b = b * a$;
- for all $a, b \in \boldsymbol{E}$, $a * b \in \boldsymbol{E}$.

NOTE 1 – If $a^0 = e$, and $a^{n+1} = a * a^n$ (for $n \geq 0$) is defined recursively, the order of $a \in \boldsymbol{E}$ is the least positive integer $n$ such that $a^n = e$.

NOTE 2 – In some cases, such as when $\boldsymbol{E}$ is the set of points on an elliptic curve, arithmetic in the finite set $\boldsymbol{E}$ is described with additive notation.

**3.2**
**cyclic group**
group $\boldsymbol{E}$ of $n$ elements that contains an element $a \in \boldsymbol{E}$, called the generator, of order $n$

**3.3**
**pairing**
function which takes two elements, $P$ and $Q$, from an elliptic curve cyclic group over a finite field, $G_1$, as input, and produces an element from another cyclic group over a finite field, $G_2$, as output, and which has the following two properties (where we assume 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 $1 \leq a \leq q - 1$, 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 $G_1$, $<P, P> \neq 1$.

**3.4**
**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

| | |
|---|---|
| $a\|\|b$ | concatenation of $a$ and $b$, in the order specified |
| $a \oplus b$ | bitwise exclusive OR of $a$ and $b$ |
| $a_1$, $a_2$ | elliptic curve coefficients |
| $(A, B, C)$ | a permutation of $(S, T_1, T_2)$, which specifies the functionality of the signature mechanisms |
| $D$ | 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; in the mechanisms based on a multiplicative group, elements of $\mathbb{E}$ are in $\mathbf{Z}_p^*$; in the mechanisms based on an additive group of elliptic curve points, elements of $\mathbb{E}$ are the points on the elliptic curve $E$ over $GF(r)$ |
| $\#\mathbb{E}$ | the cardinality of $\mathbb{E}$; in the mechanisms based on a multiplicative group, $\#\mathbb{E}$ is $p$ - 1; in the mechanisms based on an additive group of elliptic curve points, $\#\mathbb{E}$ is the number of points on the elliptic curve $E$ over $GF(r)$ |
| $gcd(N_1, N_2)$ | the greatest common divisor of integers $N_1$ and $N_2$ |
| $G$ | an element of order $q$ in $\mathbb{E}$ |
| $GF(r)$ | the Galois field of cardinality $r$ |
| $G_1$ | 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 $G_1$ (first express the data string as an integer and then convert the integer into a point on $E$ over $GF(r)$ by using the *I2P* function, see Annex C for details) |
| $h, H_2$ | 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 prime power |
| $q$ | a divisor of $\#\mathbb{E}$ and the order of $G_1$ and $G_2$, which is a prime number |
| $r$ | 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 $T$ |
| $U$ | KGC's master private key, which is a randomly chosen integer used in Mechanisms IBS-1 and IBS-2 |
| $V$ | KGC's master public key, which is an element of $G_1$ used in Mechanisms IBS-1 and IBS-2 |
| $\mathbf{Z}_N^*$ | the set of integers $U$ with $0 < U < N$ and gcd $(U, N) = 1$, with arithmetic defined modulo $N$ |
| $\alpha$ | the bit-length of a prime number (or prime power) $p$ |
| $\beta$ | the bit-length of a prime number $q$ |
| $\gamma$ | the output bit-length of hash-functions $h$ and $H_2$ |
| $\Pi_x$ | x-coordinate of $\Pi$ |
| $0_E$ | the point at infinity on the elliptic curve $E$ |
| < > | a bilinear and non-degenerate pairing |

# 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:

- $\boldsymbol{E}$ , a finite commutative group;
- $q$, a prime divisor of $\#\boldsymbol{E}$;
- $G$, an element of order $q$ in $\boldsymbol{E}$

In the group $\boldsymbol{E}$, multiplicative notation is used. It is worthwhile to note that the particular signature mechanism chosen may place additional constraints on the choice of $\boldsymbol{E}$, $q$, 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 $\boldsymbol{E}$ 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. As a result we use additive group notation here and throughout clause 7.

#### 5.1.2.2 Generation of domain parameters

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

- $G_1$, 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 $G_2$

#### 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 $Y$ is the public verification key generated from an identity string $ID$ and a hash-function $H_1$, i.e., $Y = H_1(ID)$.

### 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 $2^{80}$.

NOTE – Security level means the number of steps in the best known attack on a cryptographic primitive. If $2^{80}$ steps are required in the best known attack on a hash-function, the security level of the hash-function is $2^{80}$. For a comprehensive analysis of parameter sizes, see Silverman [27], and Lenstra and Verheul [22].

It is not necessary to select $\alpha$, $\beta$, and $\gamma$ having same security level; the security level of an implemented signature scheme is the lowest among security levels of parameters.

**Table 1 — Parameter sizes according to the security level**

| Security level | $2^{80}$ | $2^{112}$ | $2^{128}$ | $2^{192}$ | $2^{256}$ |
|---|---|---|---|---|---|
| $\alpha$ | 1024 | 2048 | 3072 | 7680 | 15360 |
| $\beta$ | 160 | 224 | 256 | 384 | 512 |
| $\gamma$ | 160 | 224 | 256 | 384 | 512 |

#### 5.1.3.2 Selecting hash-function

Selection of hash-functions should be based on those standardized in ISO/IEC 10118-3. That is, $h$ and $H_2$ are 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 $G_1$ (see Annex B for details).

NOTE 1 – The hash-functions used in this part of ISO/IEC 14888 should be collision-resistant.

NOTE 2 – The security strength for the selected hash-function must meet or exceed the security strength of the parameters in key generation. The relation between the security levels of a hash-function and parameters is shown in the above clause.

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; and
- 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; and
- 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 document.

NOTE 2 – The method of authenticating the signer is dependent on the real applications, which is out of the scope of this document.

## 5.2 Signature process

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 equation.

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

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

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 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.

### 5.2.1 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.

### 5.2.2 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, $\Pi$, 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 $E$. 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, $\Pi$.

### 5.2.3 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.

### 5.2.4 Computing the witness (the first part of the signature)

The variables to this stage are the pre-signature $\Pi$ from 5.2.2 and $M_1$ from 5.2.3. 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.

### 5.2.5 Computing the assignment

The inputs to the assignment function are the first part of the signature, which is the witness $R$ from 5.2.4, $M_2$ from 5.2.3, 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.

### 5.2.6 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 5.2.5, the permutation $(A, B, C)$ of $(S, T_1, T_2)$, a variable $D$ in 5.1.2 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 equation

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

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

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

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

The pair $(R, S)$ will be called the signature, $\Sigma$.

### 5.2.7 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.

NOTE – 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.

### 5.2.8 Constructing the signed message

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

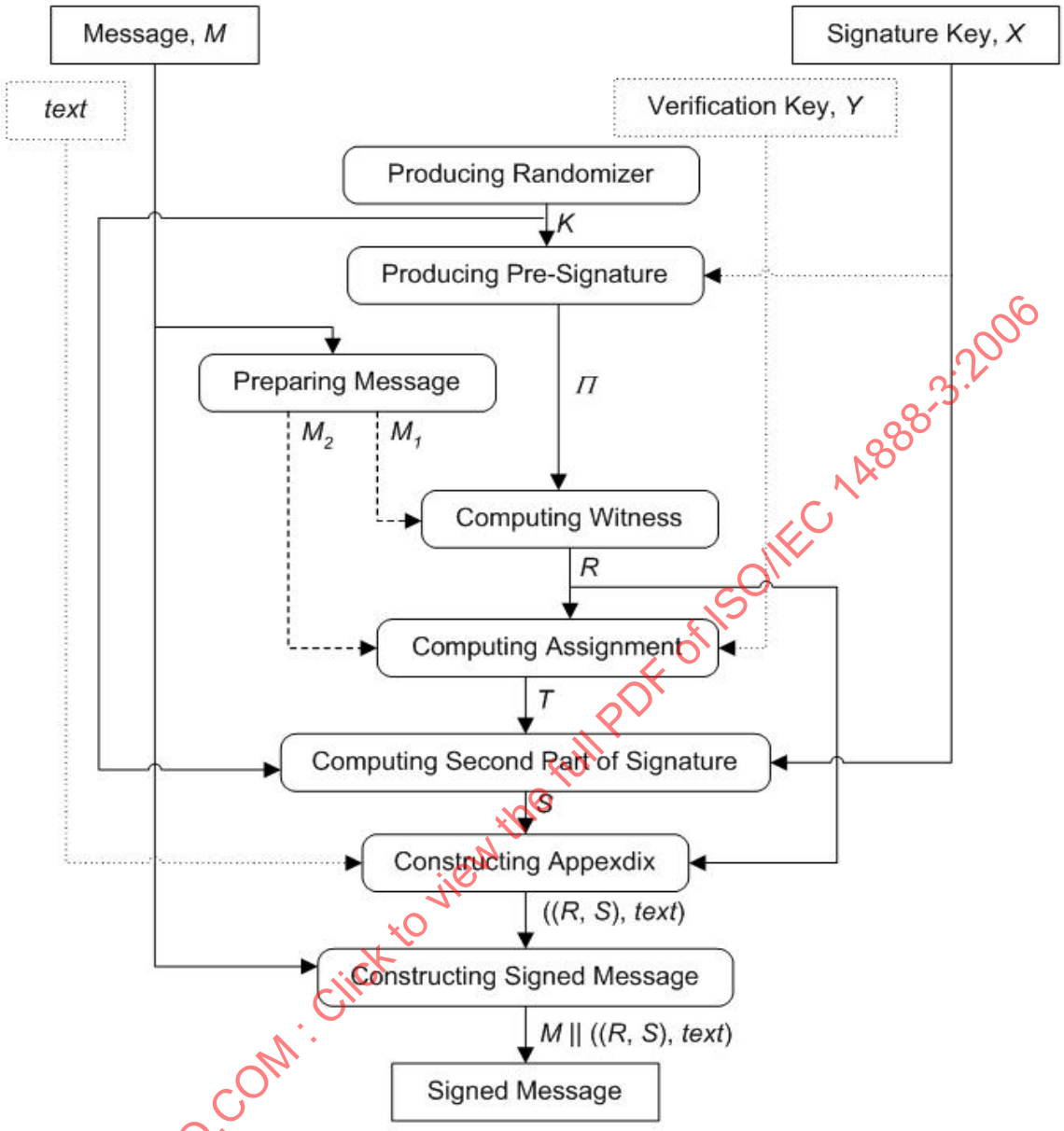




**Figure 1 — Signature process with randomized witness (one of $M_1$ and $M_2$ is $M$, the other is empty)**

### 5.3 Verification process

The verification process consists of 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.

### 5.3.1 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.

### 5.3.2 Preparing message for verification

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

### 5.3.3 Retrieving the assignment

This stage is identical to 5.2.5. The inputs to the assignment function consist of the witness $R$ from 5.3.1, $M_2$ from 5.3.2, 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 $T$.

### 5.3.4 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 5.3.3, the second part of the signature $S$ from 5.3.1, and optionally $R$ from 5.3.1. 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 $\overline{\Pi}$.

In the certificate-based mechanisms, $\overline{\Pi}$ is computed in $E$ as

$$\overline{\Pi} = Y^m G^n,$$

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.

### 5.3.5 Recomputing the witness

The computations at this stage are the same as in 5.2.4. The verifier executes the witness function. The inputs are $\overline{\Pi}$ from 5.3.4 and $M_1$ from 5.3.2. The output is the recomputed witness, $\overline{R}$.

In Mechanism IBS-2 specified in Clause 7.2, the process of recomputing the witness is computing two verification functions, instead of computing $\overline{R}$.

### 5.3.6 Verifying the witness

The signature is verified if the recomputed witness, $\overline{R}$, from 5.3.5 is equal to $R$ from 5.3.1.

In Mechanism IBS-2 specified in Clause 7.2, the process of verifying the witness involves checking whether the two verification function values computed in 5.3.5 are identical, instead of verifying whether $R = \overline{R}$ holds.

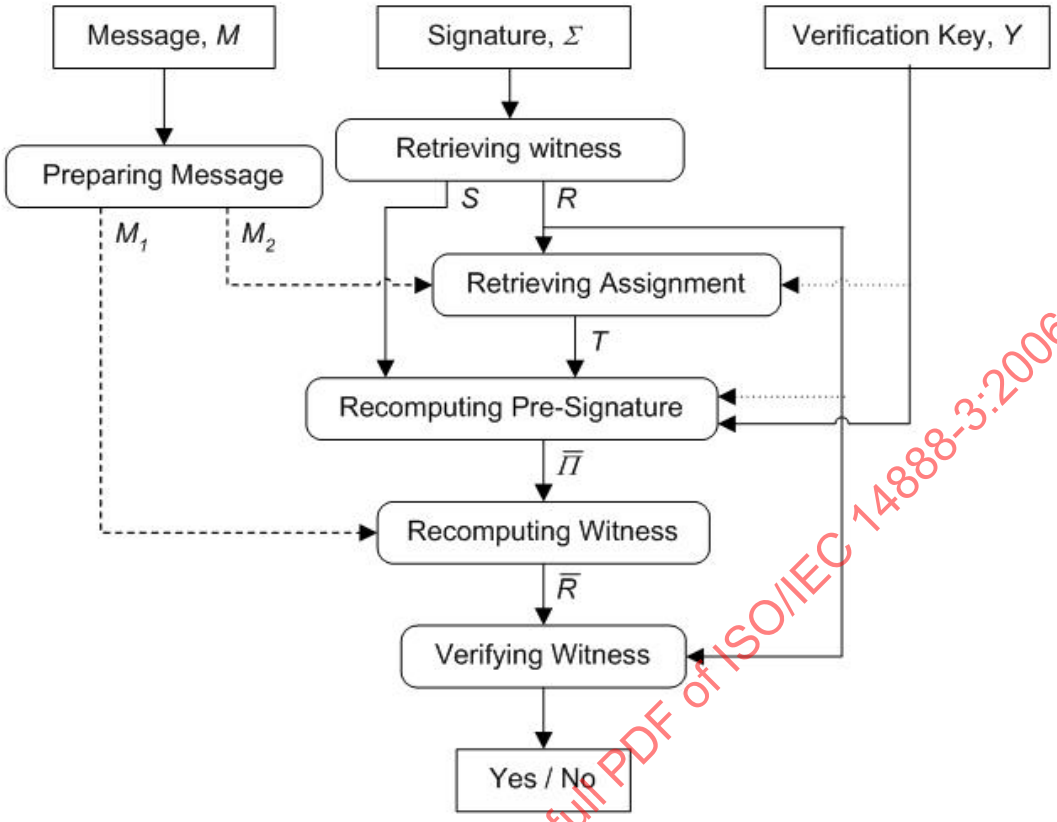


**Figure 2 — Verification process with a randomized witness**



## 6 Certificate-based mechanisms

This clause specifies six certificate-based mechanisms. These make use of arithmetic in the multiplicative group $\mathbf{Z}_p^*$ or the additive group elliptic curve points. The mechanisms using arithmetic in the multiplicative group $\mathbf{Z}_p$, are the Digital Signature Algorithm (DSA), the Korean Certificate-based Digital Signature Algorithm (KCDSA) and the Pointcheval/Vaudenay algorithm. The mechanisms using arithmetic in the additive group of elliptic curve points are the Elliptic Curve DSA (EC-DSA), the Elliptic Curve KCDSA (EC-KCDSA) and the Elliptic Curve German Digital Signature Algorithm (EC-GDSA).

In elliptic curve arithmetic, an elliptic curve point is represented as affine coordinates. That is, an elliptic curve point $\Pi$ has two coordinates: x-coordinate, $\Pi_x$, and y-coordinate, $\Pi_y$,. Elliptic curves for EC-DSA, EC-KCDSA, and EC-GDSA are restricted to non-singular curves.

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

### 6.1 DSA

DSA (Digital Signature Algorithm) is a signature mechanism with $\mathbf{E} = \mathbf{Z}_p^*$, $p$ 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

$$R = \Pi \bmod q,$$

and the assignment function by the formula

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

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

NOTE – This mechanism is taken from the U.S. National Institute of Standards and Technology (NIST) Federal Information Processing Standards Publication 186-3 (FIPS PUB 186-3). The notation here has been changed slightly from FIPS PUB 186-3 to conform with the notation used elsewhere in this part of ISO/IEC 14888.

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

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

Thus the signature equation becomes

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

### 6.1.1 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 ($\alpha$,$h$) are allowed in DSA, which are (1024, SHA-1), (2048, SHA-224), (2048, SHA-256), and (3072, SHA-256). Corresponding $\beta$ should be selected according to $\alpha$ in Section 5.1.3.1. 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. 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.

NOTE 1 – It is recommended that all users check the proper generation of the DSA public parameters according to FIPS PUB 186-3.

NOTE 2 – It is recognized that DSA possesses an unfavourable property in which an attack can be mounted where collisions on the underlying hash-functions can be found with a complexity of $2^{74}$, $2^{101}$, and $2^{114}$ as compared to $2^{80}$, $2^{112}$ and, $2^{128}$, respectively in the most secure case [30]. This attack though is easily detectable. For users who may still wish to avoid this property, it can be prevented by using the mechanisms of 6.2, 6.3 and 6.5.

NOTE 3 – SHA-1 has recently been demonstrated to provide less than 80 bits of security for digital signatures. The use of SHA-1 is not recommended for the generation of digital signatures.

### 6.1.2 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.

### 6.1.3 Signature process

#### 6.1.3.1 Producing the randomizer

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

#### 6.1.3.2 Producing the pre-signature

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

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

#### 6.1.3.3 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., $M_2 = M$.

#### 6.1.3.4 Computing the witness

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

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

#### 6.1.3.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (-R, -BS2I(H))$ where $H = h(M_2)$ is the hash-code of message $M_2$. The conversion function, $BS2I$, is given in Annex B.

#### 6.1.3.6 Computing the second part of the signature

The signature is $(R, S)$ where $R$ is computed in 6.1.3.4, and

$S = (K^{-1} (BS2I(h(M_2)) + XR)) \bmod q$.

The value of $h(M_2)$ is an $\gamma$-bit string output of the appropriate hash-function in 6.1.1. For use in computing $S$, this string shall be converted to an integer.

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).

#### 6.1.3.7 Constructing the appendix

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

#### 6.1.3.8 Constructing the signed message

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

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

### 6.1.4 Verification process

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

#### 6.1.4.1 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.

#### 6.1.4.2 Preparing the message for verification

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

#### 6.1.4.3 Retrieving the assignment

This stage is identical to 6.1.3.5. The inputs to the assignment function consist of the witness $R$ from 6.1.4.1 and $M_2$ from 6.1.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.1.3.5.

#### 6.1.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.1.4.3 and second part of the signature $S$ from 6.1.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature using the formula

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

#### 6.1.4.5 Recomputing the witness

The computations at this stage are the same as in 6.1.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from 6.1.4.4. Note that $M_1$ is empty. The output is the recomputed witness $\overline{R}$ such that $\overline{R} = \overline{\Pi} \bmod q$.

#### 6.1.4.6 Verifying the witness

The verifier compares the recomputed witness, $\overline{R}$ from 6.1.4.5 to the value of $R$ from 6.1.4.1. If $\overline{R} = R$, then the signature is valid.

### 6.2 KCDSA

KCDSA (Korean Certificate-based Digital Signature Algorithm) is a signature mechanism with $\mathbf{E} = \mathbf{Z}_p^*$, $p$ 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

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

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

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

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

NOTE – This mechanism is taken from the Korean Telecommunications Technology Association Standard (TTAS.KO-12.0001/R1), 20 Dec 2000. The notation here has been changed slightly from TTAS.KO-12.0001/R1 to conform with notation used elsewhere in this part of ISO/IEC 14888.

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

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

Thus the signature equation becomes

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

### 6.2.1 Parameters

$\alpha$ 1024+256$i$, for $i$ an integer $0 \le i \le 4$

$\beta$ 160+32$j$, for $j$ an integer $0 \le j \le 3$

$p$ 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 $\mathbf{Z}_p^*$

$I$ the input block size (in bits) of the selected hash-function $h$

Hash-function identifier or OID with specified hash-function

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

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

### 6.2.2 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.

### 6.2.3 Signature process

#### 6.2.3.1 Producing the randomizer

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

#### 6.2.3.2 Producing the pre-signature

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

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

#### 6.2.3.3 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., $M_2 = M$.

#### 6.2.3.4 Computing the witness

The signing entity computes $R = h(I2BS(\Pi))$, where the output of $H$ is the hash-code of the bit string of length $\alpha$ converted from the pre-signature $\Pi$. The witness is simply a function of the pre-signature. Thus,

$$R = h(I2BS(G^K \bmod p)).$$

To compute $R$, an integer $\Pi$ shall be converted to a bit string of length $\alpha$. The conversion rule, *I2BS*, is given in Annex B.

#### 6.2.3.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (V, -1)$ where $V = BS2I(R \oplus H) \bmod q$, where $H=h(Y' || M_2)$ is the hash-code of the concatenation of $Y' = I2BS(Y \bmod 2^l)$ 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 $q$.

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

#### 6.2.3.6 Computing the second part of the signature

The signature is $(R, S)$ where $R$ is computed in 6.2.3.4, and

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

#### 6.2.3.7 Constructing the appendix

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

#### 6.2.3.8 Constructing the signed message

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

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

### 6.2.4 Verification process

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 14888-1:1998, clause 9 for additional required data items).

#### 6.2.4.1 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 the bit length of $R$ is equal to the output bit length of the employed hash-function $h$ and $0 < S < q$. If either condition is violated the signature shall be rejected.

#### 6.2.4.2 Preparing the message for verification

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

#### 6.2.4.3 Retrieving the assignment

This stage is identical to 6.2.3.5. The inputs to the assignment function consist of the witness $R$ from 6.2.4.1 and $M_2$ from 6.2.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.2.3.5.

#### 6.2.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.2.4.3 and second part of the signature $S$ from 6.2.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature using the formula

$$\overline{\Pi} = Y^{S \bmod q} \; G^{T_1 \bmod q} \bmod p.$$

#### 6.2.4.5 Recomputing the witness

The computations at this stage are the same as in 6.2.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from 6.2.4.4. Note that $M_1$ is empty. The output is the recomputed witness $\overline{R}$.

#### 6.2.4.6 Verifying the witness

The verifier compares the recomputed witness, $\overline{R}$ from 6.2.4.5 to the value of $R$ from 6.2.4.1. If $\overline{R} = R$, then the signature is valid.

### 6.3 Pointcheval/Vaudenay algorithm

The method of Pointcheval/Vaudenay is a variant of the DSA algorithm, with $\mathbf{E} = \mathbf{Z}_p^*$, $p$ 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 $H = h(I2BS(R)||M)$ 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.

NOTE – This mechanism is based on the algorithm designed by D. Pointcheval and S. Vaudenay in [26].

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

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

Thus the signature equation becomes

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

#### 6.3.1 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

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

### 6.3.2 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.

### 6.3.3 Signature process

#### 6.3.3.1 Producing the randomizer

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

#### 6.3.3.2 Producing the pre-signature

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

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

#### 6.3.3.3 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., $M_2 = M$.

#### 6.3.3.4 Computing the witness

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

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

#### 6.3.3.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (-R, -BS2I(H))$, where $H = h(I2BS(R)||M_2)$ 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 $|p|$.

#### 6.3.3.6 Computing the signature

The signature is $(R, S)$ where $R$ is computed in 6.3.3.4, and

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

#### 6.3.3.7 Constructing the appendix

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

#### 6.3.3.8 Constructing the signed message

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

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

#### 6.3.4 Verification process

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 14888-1:1998, clause 9 for additional required data items).

##### 6.3.4.1 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.

##### 6.3.4.2 Preparing the message for verification

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

##### 6.3.4.3 Retrieving the assignment

This stage is identical to 6.3.3.5. The inputs to the assignment function consist of the witness $R$ from 6.3.4.1 and $M_2$ from 6.3.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.3.3.5.

##### 6.3.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.3.4.3 and second part of the signature $S$ from 6.3.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature by computing it using the formula

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

##### 6.3.4.5 Recomputing the witness

The computations at this stage are the same as in 6.3.3.4. The verifier executes the witness function. The inputs are $\overline{\Pi}$ from 6.3.4.4. Note that $M_1$ is empty. The output is the recomputed witness $\overline{R}$.

##### 6.3.4.6 Verifying the witness

The verifier compares the recomputed witness, $\overline{R}$, from 6.3.4.5 to the value of $R$ from 6.3.4.1. If $\overline{R} = R$, then the signature is valid.

### 6.4 EC-DSA

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 equation are set as follows

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

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

NOTE – This mechanism is based on the algorithm described in [5].

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

$$R = FE2I(\Pi_x) \bmod q.$$

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

Thus the signature equation becomes

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

### 6.4.1 Parameters

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

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

NOTE 1 – It is recommended that all users check the proper generation of the EC-DSA public parameters according to X9.62 [5].

NOTE 2 SHA-1 has recently been demonstrated to provide less than 80 bits of security for digital signatures. The use of SHA-1 is not recommended for the generation of digital signatures.

### 6.4.2 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.

### 6.4.3 Signature process

#### 6.4.3.1 Producing the randomizer

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

#### 6.4.3.2 Producing the pre-signature

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

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

#### 6.4.3.3 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., $M_2 = M$.

#### 6.4.3.4 Computing the witness

The signing entity computes $R = FE2I(\Pi_x) \bmod q$.

#### 6.4.3.5 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 $\lceil \log_2 q \rceil$ 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 $(-R, -BS2I(H))$.

#### 6.4.3.6 Computing the second part of the signature

The signature is $(R, S)$ where $R$ is computed in 6.4.3.4, and

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

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).

#### 6.4.3.7 Constructing the appendix

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

#### 6.4.3.8 Constructing the signed message

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

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

### 6.4.4 Verification process

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

#### 6.4.4.1 Retrieving the witness

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

#### 6.4.4.2 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 $M_2 = M$.

#### 6.4.4.3 Retrieving the assignment

This stage is identical to 6.4.3.5. The inputs to the assignment function consist of the witness $R$ from 6.4.4.1 and $M_2$ from 6.4.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.4.3.5.

#### 6.4.4.4 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.4.4.3 and second part of the signature $S$ from 6.4.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature by computing it using the formula

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

#### 6.4.4.5 Recomputing the witness

The computations at this stage are the same as in 6.4.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from 6.4.4.4. The output is the recomputed witness $\overline{R}$.

#### 6.4.4.6 Verifying the witness

The verifier compares the recomputed witness, $\overline{R}$ from 6.4.4.5 to the retrieved version of $R$ from 6.4.4.1. If $\overline{R} = R$, then the signature is verified.

### 6.5 EC-KCDSA

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

$$R = h(FE2BS(\Pi_x)).$$

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

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

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

NOTE – This mechanism is taken from the Korean Telecommunications Technology Association Standard (TTAS.KO-12.0015), 19 Dec 2001. The notation here has been changed slightly from TTAS.KO-12.0015 to conform with notation used elsewhere in this part of ISO/IEC 14888.

NOTE – The domestic standard for EC-KCDSA has been revised. This is the reason of the slight difference between the definition of EC-KCDSA in this document and the definition in 15946-2.

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

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

Thus the signature equation becomes

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

#### 6.5.1 Parameters

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

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

### 6.5.2 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.

### 6.5.3 Signature process

#### 6.5.3.1 Producing the randomizer

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

#### 6.5.3.2 Producing the pre-signature

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

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

#### 6.5.3.3 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., $M_2 = M$.

#### 6.5.3.4 Computing the witness

The signing entity computes $R = h(FE2BS(\Pi_x))$, where output of $h$ is the hash-code of $\Pi_x$. The witness is simply a function of the pre-signature. To compute $R$, the field element $\Pi_x$ shall be converted to a bit string. The conversion rule, *FE2BS*, is given in Annex B.

#### 6.5.3.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (V, -1)$ where $V = BS2I(R \oplus H) \bmod q$, 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 $FE2BS(Y_x)||FE2BS(Y_y)$. In computing $V$, the bit string $R \oplus H$ shall be converted to an integer before modulo reduction with respect to $q$.

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

#### 6.5.3.6 Computing the second part of the signature

The signature is $(R, S)$ where $R$ is computed in 6.5.3.4, and

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

#### 6.5.3.7 Constructing the appendix

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

#### 6.5.3.8 Constructing the signed message

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

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

#### 6.5.4 Verification process

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

The verifier also acquires the necessary data items for the verification process: for example, the verification key $Y$ (see ISO/IEC 14888-1:1998, clause 9 for additional required data items).

##### 6.5.4.1 Retrieving the witness

The verifier retrieves the witness $R$ and the second part of the signature $S$ from the appendix. And, the verifier first checks to see that the bit length of $R$ is equal to the output bit length of the employed hash-function $h$ and $0 < S < q$. If either condition is violated the signature shall be rejected.

##### 6.5.4.2 Preparing the message for verification

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

##### 6.5.4.3 Retrieving the assignment

This stage is identical to 6.5.3.5. The inputs to the assignment function consist of the witness $R$ from 6.5.4.1 and $M_2$ from 6.5.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.5.3.5.

##### 6.5.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.5.4.3 and the second part of the signature $S$ from 6.5.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature using the formula

$$\overline{\Pi} = [S \bmod q]Y + [T_1 \bmod q]G.$$

##### 6.5.4.5 Recomputing the witness

The computations at this stage are the same as in 6.5.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from 6.5.4.4. Note that $M_1$ is empty. The output is the recomputed witness $\overline{R}$.

##### 6.5.4.6 Verifying the witness

The verifier compares the recomputed witness $\overline{R}$ from 6.5.4.5 to the value of $R$ from 6.5.4.1. If $\overline{R} = R$, then the signature is valid.

### 6.6 EC-GDSA

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

$$R = FE2I(\Pi_x) \bmod q.$$

The coefficients $(A, B, C)$ of the EC-GDSA signature equation 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 equation becomes

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

NOTE – EC-GDSA stands for Elliptic Curve German Digital Signature Algorithm (H. Erwin, and S. Pascale. "Digital Signature Scheme EC-GDSA," German Federal Office for Information Security, December 2005).

### 6.6.1 Parameters

| | |
|---|---|
| $\boldsymbol{F}$ | a finite field |
| $\boldsymbol{E}$ | an elliptic curve group over field $\boldsymbol{F}$ |
| $\#\boldsymbol{E}$ | the cardinality of $\boldsymbol{E}$ |
| $q$ | a prime divisor of $\#\boldsymbol{E}$ |
| $G$ | 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.

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

### 6.6.2 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.

### 6.6.3 Signature process

#### 6.6.3.1 Producing the randomizer

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

#### 6.6.3.2 Producing the pre-signature

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

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

#### 6.6.3.3 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., $M_2 = M$.

#### 6.6.3.4 Computing the witness

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

#### 6.6.3.5 Computing the assignment

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

#### 6.6.3.6 Computing the second part of the signature

The signature is $(R, S)$ where $R$ is computed in 6.6.3.4, and

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

#### 6.6.3.7 Constructing the appendix

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

#### 6.6.3.8 Constructing the signed message

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

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

### 6.6.4 Verification process

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

#### 6.6.4.1 Retrieving the witness

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

#### 6.6.4.2 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 $M_2 = M$.

#### 6.6.4.3 Retrieving the assignment

This stage is identical to 6.6.3.5. The inputs to the assignment function consist of the witness $R$ from 6.6.4.1 and $M_2$ from 6.6.4.2. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, 6.6.3.5.

#### 6.6.4.4 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key $Y$, assignment $T = (T_1, T_2)$ from 6.6.4.3 and second part of the signature $S$ from 6.6.4.1. The verifier obtains a recomputed value $\overline{\Pi}$ of the pre-signature by computing it using the formula

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

#### 6.6.4.5 Recomputing the witness

The computations at this stage are the same as in 6.6.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from 6.6.4.4. The output is the recomputed witness $\overline{R}$.

#### 6.6.4.6 Verifying the witness

The verifier compares the recomputed witness, $\overline{R}$ from 6.6.4.5 to the retrieved version of $R$ from 6.6.4.1. If $\overline{R} = R$, then the signature is verified.

## 7 Identity-based mechanisms

This clause specifies two identity-based mechanisms that are based on pairings. An identity-based signature mechanism involves three entities: a trusted Key Generation Centre (KGC), a signer and a verifier.

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 $\Sigma$
- hash-function identifier for $H_1$ and $H_2$ (optional)

NOTE – 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 – 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 $\geq 2$ and $m$ is an integer $\geq 1$.

### 7.1 IBS-1

NOTE – This mechanism is based on the algorithm designed by Hess in [19].

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 equation becomes

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

### 7.1.1 Parameters

The signature mechanism takes place in an environment where the entities involved share the following parameters, which have been defined in Clause 4.1: $G_1$, $G_2$, $P$, $q$, < >, $H_1$, and $H_2$.

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

### 7.1.2 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.

### 7.1.3 Signature process

#### 7.1.3.1 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.

#### 7.1.3.2 Producing the pre-signature

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

$\prod = \langle X, P \rangle^K$.

NOTE – $\prod = \langle X, P \rangle^K$ is an element in $G_2$ over $GF(p^m)$ where $p$ is a prime and $m$ is an integer. In the existing elliptic curves used to implement this mechanism, when $p = 2$, $m = 4$, when $p = 3$, $m = 6$, and when $p > 3$, $m = 2$.

#### 7.1.3.3 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., $M_1 = M$.

#### 7.1.3.4 Computing the witness

The signer applies the hash-function $H_2$ to $M_1||FE2BS(\prod)$ (concatenation of $M_1$ and $FE2BS(\prod)$) to obtain the witness

$R = H_2(M_1||FE2BS(\prod))$.

NOTE – In the existing elliptic curves used to implement this mechanism, $\prod$ is an element in $G_2$ over $GF(p^m)$ where $p$ is a prime and $m$ is an integer. When $p > 3$ and $m = 2$, $\prod$ can be represented as $\prod_a + sqrt(-1)\prod_b$ where $\prod_a$ and $\prod_b$ are the elements in $GF(p)$. In this case $R = H_2(M_1||FE2BS(\prod))$ is possibly computed as $H_2(M_1||FE2BS(\prod_a)||FE2BS(\prod_b))$, e.g., in Annex F.7.

#### 7.1.3.5 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.

#### 7.1.3.6 Computing the second part of the signature

The signer computes the second part of the signature as

$S = [K - R]X$.

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

#### 7.1.3.7 Constructing the appendix

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

#### 7.1.3.8 Constructing the signed message

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

### 7.1.4 Verification process

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

#### 7.1.4.1 Retrieving the witness

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

The verifier then checks if the length of $R$ is equal to the output length of $H_2$ and $S \in G_1$ hold; if either condition is violated the signature shall be rejected. Otherwise the verifier carries out in the follows steps.

#### 7.1.4.2 Preparing message for verification

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

#### 7.1.4.3 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.

#### 7.1.4.4 Recomputing the pre-signature

The verifier recomputes the pre-signature value

$\overline{\Pi} = \langle S, P \rangle * \langle Y, V \rangle^R$.

NOTE – The pairing $\langle Y, V \rangle$ can be pre-computed.

#### 7.1.4.5 Recomputing the witness

The verifier recomputes the witness

$$\overline{R} = H_2(M_1 || FE2BS(\overline{\Pi}))$$.

#### 7.1.4.6 Verifying the witness

The verifier checks whether $\overline{R} = R$ holds. If it holds, the signature is verified; otherwise, it is invalid.

## 7.2 IBS-2

NOTE – This mechanism is based on the algorithm designed by Cha and Cheon in [11].

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 equation becomes

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

### 7.2.1 Parameters

The parameters are the same as in Clause 7.1.1.

### 7.2.2 Generation of master key and signature/verification key

This operation is the same as in Clause 7.1.2.

### 7.2.3 Signature process

#### 7.2.3.1 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.

#### 7.2.3.2 Producing the pre-signature

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

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

#### 7.2.3.3 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., $M_2 = M$.

#### 7.2.3.4 Computing the witness

The signer obtain the witness from the pre-signature result

$$R = \Pi.$$

#### 7.2.3.5 Computing the assignment

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

$T_1$ = -$Y$, and

$T_2$ = [- $H$]$Y$

where $H = H_2(M_2||FE2BS(\Pi_x))$. However, the signer only needs to compute the value of $H$.

#### 7.2.3.6 Computing the second part of the signature

The signer computes the second part of the signature as,

$S = [K + H]X$.

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

#### 7.2.3.7 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*).

#### 7.2.3.8 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*).

### 7.2.4 Verification process

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

#### 7.2.4.1 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.

#### 7.2.4.2 Preparing message for verification

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

#### 7.2.4.3 Retrieving the assignment

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

$T_1$ = -$Y$, and

$T_2$ = [-$H$]$Y$.

where $H = H_2(M_2||FE2BS(R_x))$. However the verifier only needs to recompute the value of $H$.

#### 7.2.4.4 Recomputing the pre-signature

The verifier retrieves the pre-signature value as

$\overline{\Pi} = R$.

#### 7.2.4.5 Recomputing the witness

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

#### 7.2.4.6 Verifying the witness

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

# Annex A
(normative)

# ASN.1 module

This annex lists the ASN.1 module assigned to the digital signature mechanisms specified in this part of ISO/IEC 14888.

```
DigitalSignatureWithAppendixDL {
   iso(1) standard(0) digital-signature-with-appendix (14888) part(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) part(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-with-sha1(3) }
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-SHA1(1) }
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) }

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,
```

```
   ...  -- 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
}

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 OIDs for EC-DSA presented in X9.62 are as follows:
{iso(1) member-body(2) us(840) ansi-x9-62(10045) signatures(4)
ecdsa-with-Recommended(2)} - EC-DSA with Recommended

```
{iso(1) member-body(2) us(840) ansi-x9-62(10045) signatures(4)
    ecdsa-with-Specified(3)}
```
- EC-DSA with Specified

NOTE 4 - 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*)

A field element in ***F*** can be converted to integer by the following function:

- In case of *GF*(p), then $FE2I(\Pi_x) = \Pi_x$.
- In case of $GF(2^m)$, then $\Pi_x$ is a bit string of length *m*. Let $s_{m-1}s_{m-2}\cdots s_0$ be the bit string $\Pi_x$. Then:

$$FE2I(\Pi_x) = \sum_{i=0}^{m-1} 2^i s_i$$

- In case of $GF(p^m)$, then $\Pi_x$ is a *p*-ary string of length *m*. Let $s_{m-1}s_{m-2}\cdots s_0$ be the *p*-ary string $\Pi_x$. Then:

$$FE2I(\Pi_x) = \sum_{i=0}^{m-1} p^i s_i$$

## B.2 Conversion from an integer to a field element (*I2FE*)

An integer *x* can be converted to a field element by the following function:

- In case of *GF*(*p*), then *I2FE*(*x*) = *x*.
- In case of $GF(2^m)$, then *I2FE*(*x*) = *I2BS*(*x*).
- In case of $GF(p^m)$, *x* can be expressed as $x = x_0 + x_1 * p + \ldots + x_{m-1} * p^{m-1}$.
  Then $I2FE(x) = \{x_{m-1},\cdots,x_1,x_0\}$.

## B.3 Conversion from a field element to a bit sequence (*FE2BS*)

A field element can be converted to a bit sequence by the following function:

*FE2BS*(*a*) = *I2BS*(*FE2I*(*a*))

- In case of *GF*(*p*), then the bit length is $8 * \lceil log_{256}\, p \rceil$.
- In case of $GF(2^m)$, then the bit length is $8 * \lceil m/8 \rceil$.
- In case of $GF(p^m)$, then the bit length is $8 * \lceil log_{256}\, p^m \rceil$.

## B.4 Conversion from a bit sequence to an integer (*BS2I*)

A *g*-long sequence of bits $\{x_1, \ldots, x_g\}$ is converted to an integer by the rule

$\{x_1, \ldots, x_g\} \rightarrow x_1 * 2^{g-1} + x_2 * 2^{g-2} + \ldots + x_{g-1} * 2 + x_g$.

## B.5 Conversion from an integer to a bit sequence (*I2BS*)

An integer $x$ in the range $0 \le x < 2^g$ may be converted to a $g$-long sequence of bits by using its binary expansion as shown below:

$$x = x_1 * 2^{g-1} + x_2 * 2^{g-2} + \ldots + x_{g-1} * 2 + x_g \rightarrow \{x_1, \ldots, x_g\}.$$

## B.6 Conversion between an integer and an octet string (*I2OS* & *OS2I*)

This section specifies Functions *I2OS* (Integer to Octet String conversion) and *OS2I* (Octet String to Integer conversion).

Function *I2OS* takes as input a non-negative integer $x$, and produces the unique octet string $M_{l-1}M_{l-2} \ldots M_0$ of length $l$ as output, where $l = \lceil \log_{256}(x + 1) \rceil$ is the length in octets of $x$. *I2OS* is defined as follows:

1. Write $x$ in its unique $l$-digit base 256 representation:

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

   where $0 \le x_i < 256$.

2. Let the octet $M_i$ have the value $x_i$ for $0 \le i \le l-1$.

3. Output the octet string $M_{l-1}\, M_{l-2} \ldots M_0$.

For example, *I2OS*(10945) = 2A C1.

Function *OS2I* takes an octet string $M_{l-1}\, M_{l-2} \ldots M_0$ as input and produces a non-negative integer $y$ as output. It is defined as follows:

1. Let integer $y_i$ have the value of the octet $M_i$ for $0 \le i \le l-1$.

2. Compute the integer $y = y_{l-1}\, 256^{l-1} + y_{l-2}\, 256^{l-2} + \ldots + y_1\, 256 + y_0$.

3. Output $y$.

For example, *OS2I*(2A C1) = 10945.

Note that the octet string of length zero (the empty octet string) is converted to the integer 0 and vice versa.

# Annex C
(informative)

# Conversion functions (II)

## C.1 Conversion from an integer to a point (*I2P*)

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

NOTE – 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 $T$ 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.

1. Set $v = BS2I(H_2(I2OS(u)))$ mod $q$. If $v = 0$, output "invalid" and stop.

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

3. 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:

   (a) Let $x$ have the value of $v$.

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

   (c) 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.

      1) 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.

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

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

      NOTE – The operation of finding a field element $d$ such that $d^2 = c$ mod $p$ is given in [5] and [20].

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

   (e) Set point $T = (x, y)$ and output $T$.

4. 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:

   (a) Set $x = I2FE(v)$.

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

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

(d) 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 1.

NOTE – The operations of determining the existence of and finding a square root of a field element are given in [5] and [20].

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

(f) Set point $T = (x, y)$ and output $T$.

5. If $r$ is an even prime power ($r = 2^m$, $m \geq 2$) 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:

(a) Set $x = I2FE(v)$.

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

(c) 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 1.

NOTE – 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 [5] and [20].

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

(e) Set point $T = (x, y)$ 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 2*q* as described below. Conversion functions between integer and sequence are given in Annex B.

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

Step 1. Choose an arbitrary sequence of at least $\beta$ 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, $q = U \text{ OR } 2^{\beta-1} \text{ OR } 1$. 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

$$V_k = h((SEED + offset + k) \bmod 2^s).$$

Step 8. Let *W* be the integer $W = V_0 + V_1 * 2^m + \ldots + V_{n-1} * 2^{(n-1)*m} + (V_n \bmod 2^b) * 2^{n*m}$ and let $X = W + 2^{\alpha-1}$. Note that $0 \le W < 2^{\alpha-1}$ and hence $2^{\alpha-1} \le X < 2^{\alpha}$.

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

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 $counter \ge 2^{12} = 4096$ go to Step 1, otherwise (i.e., if *counter* < 4096) go to Step 7.

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

NOTE – This procedure is taken from FIPS 186-3, 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. $e = (p-1)/q$.

Step 2. Set $F$ = any integer, where $1 < F < p-1$, 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* and *SEED*. Let h be the appropriate hash-function for the $(\alpha, \beta)$ 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*.

Step 1. $e = (p-1)/q$.

Step 2. *count* = 1.

Step 3. $U$ = *SEED* || "ggen" || *index* || *count*.

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

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

Step 6. If $G = 1$, then increment *count* and go to step 3.

# Annex E
(informative)

# The Weil and Tate pairings

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 [24, 28] and a modified version [7] and a modified version of Tate pairing [16, 17]. In this document, $< >$ 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 $1 \le a \le q - 1$,
  $<P_1 + P_2, Q> = <P_1, Q> * <P_2, Q>$,
  $<P, Q_1 + Q_2> = <P, Q_1> * <P, Q_2>$, and
  $<[a]P, Q> = <P, [a]Q> = <P, Q>^a$.
- Non-degenerate: If $P$ is a non-identity point of the cyclic group, $<P, P> \neq 1$.

## E.1 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 equation $y^2 + a_1*x*y + a_3*y = x^3 + a_2*x^2 + a_4*x + a_6$.

— 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 an even prime power ($r = 2^m$, $m \geq 2$) 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.

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

   Set $r = 1$

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

   $f_r = f_r * f_r * g(A, A, C)$;

   $A = A + A$;

   Set $r = 2 * r$

   if $n_i \neq 0$ then {

   $f_r = f_r * g(A, D, C)$;

   $A = A + D$;

   Set $r = r + 1$

   }

   }

3. Set $d(D, C, l) = f_r$ and output $d(D, C, l)$.

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

## E.2 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 $<P, Q> = ( d(Q, P + T) / d(Q, T) ) / ( d(P, Q - T) / d(P, -T) )$.

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

## E.3 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 [6, 18, 35].

# Annex F
(informative)

# Numerical examples

In this annex, unless otherwise explicitly stated, all the values are expressed in hexadecimal notation.

## F.1 DSA mechanism

### F.1.1 Example 1

A complete explanation of the generation of all values is given in Appendix 5 of [14, 15]. This example is sample value for DSA with $\alpha$ = 2048 and $\beta$ = 224. All hashing, including generation of domain parameters, is performed with SHA-224.

#### F.1.1.1 Parameters

$\alpha$ = 2048

SEED = 61215F51 1B357887 16FC455A 9BD06ABD ED534B9E 52FDB046 9A6728A6

*F* = 2

*P* = 9441A6EC 7F1E92B5 C84800E4 7E72A46C 7EFBB7B5 CBFADFC7 ECEF7EC6
559F5CD8 21936BA2 A81F0323 84C2C1BD 69F8C322 B730D79C B9979370
A05F9047 58308F43 9D43D6F2 79F7B138 32A250B0 7B4BE08C 3A01642F
6D76AEE2 DE22F086 DA70AB32 802585D6 47E95A65 B962A924 FDA568CC
0D199886 3B3E302C CAA20359 A1BC0DD0 6601410A 5594386E 692E8450
5FBD2D24 775862 5C DE456646 3004F2DE 9C042AA2 F0B38034 EA246F91
00B03FFF 8786441D CE231FFE 7CA3A446 8C5480F1 C68B5D81 94F6976A
287F1279 EAFFB979 A0543B5A 5E7638FE 74AF0EC2 AA44CEB4 C30C2DB2
4DB042D4 14F31938 8BCF1288 C66A4590 A942D5EF 2D481313 A0E90DF2
B6A6F341

*Q* = E7EDAB1B C6A00E83 0498C51B D648964A DE55C198 01790BD3 144B1D2D

*Counter* = 105

*G* = 834ED7D6 10F6E78D 0B55ADA8 AA9ECE03 B8F17F21 6F076735 30103BB6
4709425F 864E0928 FFCA0DC8 9DE25527 05B144F9 4A54EB19 AEDAF4DE
756BF65C FB6BD202 F71E4E51 9C31647A 32BE00D7 89271CB9 C449CDF3
DDB4CA3D B318D2B6 C105E5F1 FF995F0A C9544A35 9B5315A8 6C8D0377
93CE0AE3 AE687FB5 2931A9CB 8AAD60BE 7D2749E7 D131FDFC CA60E0D0
4D197101 F0898F75 68178667 CE88817C BE9ACCE6 24932817 FF3729A2
7349FCD6 755F40E7 F7B2DF51 636886E4 D2AA6C8B 82920266 DFB4A9FA
4C43020C C2C69793 009571AA 51C4DDE6 AEFAE490 66B37C71 06DEF3F8
98A7C684 F7C60199 1930D39E D006D545 D49B6261 D7F786FC 1F51C110
4E4D1E96

#### F.1.1.2 Signature key and verification key

$X$ = 1DFEA14C 48027FC4 50B8D2D0 0CAD5F05 33C11BFA F8268CFD 28D5E9F6

$Y$ = 09E62BFB A85B9B1D 86838D29 AAB8882E 9215FB0D 471C0DE4 3DB0AECC
E5368AAA 83EDA4B7 F018AFA4 289939F9 6A54486F A10AA2F6 BB7C5B45
34FE8A51 6123309A FCE3BB0C 7C4845DA 02E15E4E 33EFA6A6 857EACA8
42FC04CB B5A785A2 533743AE 6BF6038A B5CDF486 C3C9BEA6 F28BF90D
EEB1F8FC 50F9FDB7 6F5A6F5A F87C64D2 22607022 F89F6978 CBA01CF8
B139B70E 0BF2464B 2562382C 5BC5C29D BADA62B3 06FE6320 967364E6
3F5F3805 404343EF F167DD67 3E4731F1 CC8DDEEF 3CD46CAA 5E3EFBC1
605673CC 4F7ECE45 38BB4B5D BCFD50DA 613C6A36 944816FC A8441BA5
A9B78A59 BE0C500B 5D5AAF9E 5BB8E902 ED3A8657 F6AE8A4B D1CCAC45
0FCBDBC7

#### F.1.1.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 1DFEA14C 48027FC4 50B8D2D0 0CAD5F05 33C11BFA F8268CFD 28D5E9F6

$h(M)$ = 23097D22 3405D822 8642A477 BDA255B3 2AADBCE4 BDA0B3F7 E36C9DA7

#### F.1.1.4 Signature

$R$ = 7D38D86E 4AE38D96 E3607C04 E87CD4FD EF3F9C61 4C2B3EF2 1A2C240D

$S$ = C50A164F 32F0D624 4154D233 1C616FFF E120054C 3D87ED5D 10396A61

#### F.1.1.5 Verification

$\overline{R}$ = 7D38D86E 4AE38D96 E3607C04 E87CD4FD EF3F9C61 4C2B3EF2 1A2C240D

### F.1.2 Example 2

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.1.2.1 Parameters

$\alpha$ = 3072

SEED = C3E4D6E8 D8339C83 E410E432 B41B286E 65AC858B EBCAB2BE 34517C59
957179CB

$F$ = 2

$P$ = 8A6F9114 3C563196 D8123824 9A8E468A 66C2896F 2B10C26C E24104AB
DBBE4029 1152C7D6 401FD703 8A97F781 15B5CF38 E1464545 F1679BFB
A31BCF3F 68BC54C0 0CB8064A 0F4BDF1C 23D47A6B 3BBB8B6A 9C664500
3C41AD94 5562EF3C 491D8693 C1344A07 7221966C F8580380 02467D4F
F3BC5AA6 7525F325 933C01C2 4E068270 BF33FBF6 E5A4891B CE4B6C15
C31C3032 9599550F 24E32C83 F63687C2 6220E96F 6C9B83FC DC19C22B

827A690C B93509C5 695012CE 6216EAF6 3F91097B 88E860B1 B3882A93
99BB96BC 3D05BC66 21C51DC8 244C7B80 6EB63635 EB695EED 2A5900E0
04317612 7787793B 5C4E3A42 F2A2E681 F7E00999 D1277BAB 6FE020DB
B7CA25DA 8A59FA6D A6B3E958 531F5519 9205D496 0748D093 F2CC3348
CE40412F A1F261C4 036F77CF F7BCC371 AB76C5F6 AB850F7D 5CDB1FE3
0779317E BC8C76A5 B06DDC28 2C088109 656FCD05 238F8D3E 7128F18D
0765E796 8E5A2A81 CD0376B7 5683D739 52357324 65EA52DA 14330D0D
F293FFCD 1A569BFC CBAF3DF4 918AAAB8 EF523399

*Q* = BEFC780B 3934C22C 6D282383 F22DB63F 472EA64B B27045A5 927E1845
6030B2D3

*Counter* = 635

*G* = 732171BE 1CBD6272 C80C3BB1 F872600E FF92928B AC9933D7 854C7BB9
F966B97A 0AC002B8 B47D86A1 0DCC65D8 D086004A CE8AF55C 11CCBC8B
DE4CCE61 45E426F1 3B32A3F0 0E0DB800 6F3EF156 E1614F23 6CEDB380
68329AA8 E872DC27 1F2FF246 D6CE55FA 259E5C83 8247E290 1070DFEE
4F30599D B26E8538 88896CDC B9163684 10E2B527 BFF3AF07 F506B977
F0F655A6 C68BF014 2BABFBFC E77A04E0 C203B6EB 83D4D4A0 4B8F6A74
50A21451 2A1E82BF 0998B555 4B4DB3C1 41BB0E4B 1C611724 27A0CB16
3ECF96B8 3429A05D FA951048 0A921D7A 4FA6F0C1 039A9EEB 4156E2E4
23A28A17 6417E596 4D41B94F E47F7800 169D7232 4399C292 90DCAD3D
EE3E6F4F 18F57B6F 31DF51B4 68BAE637 4BCC9131 AC333C5D BC5939B8
6849A2AC 7A67343D 7EB0C51B E71AEC70 5E51D67A 8937F463 23B1D651
82A1265A 914578AE DF161828 40317BEB 9B802B31 01DD5DA5 62BFF5A1
45B976F9 72D68728 87C7A135 879ED9A4 AE9D52CF B9610237 E9F00B8A
FD9E06B0 9EC152F9 D05A05F8 CB64B90E 8A8C5B4A

### F.1.2.2 Signature key and verification key

*X* = 50936A94 88B7C84E 74742189 3B6C20B2 AF6D55F2 A746B3D9 D80AB02D
C8378A11

*Y* = 72C419DD 4604D3D7 DB65244B 45686343 62C4BE9D 8A2F7C53 099FD60E
B61D3D14 D33E02EB 11A6321D 27B7B661 C4F3F921 EA8E79A0 E03F9DE2
9D77F192 4A801DFF 3200A224 A136DC70 03470F78 E669E551 F9AB0100
F2C38D29 247C5FD6 7246FA0C 712232E6 B4498CD4 A990E429 1B1914D2
F31BBD4C 19418BFB BEF1C534 86C8872B 1A34AB88 25849F3A 929D517F
68D880AB 367E2D81 0640F4B4 1F76B8F8 D38FE670 4900F9BD B83345F0
E0AE2D15 18DBFC80 EE78D318 9CAD5EDA 58BD8260 28128D3D 20ED6B05
8F493A62 8355FCB8 FC55BFF7 F9622B22 5FC3E6B4 7AC4884B 08179AFE
0DBFAAC6 633C0049 92938E5B CAB14D8A C21F0117 6546EB64 4BAE2324
8CE5FC6C 1E515441 FB4EAE14 42907511 F4662BA1 352E791E BE872AB6
6405F48B 67BD6151 E0B8C8E8 CDF45682 7F7C2007 0B01B14C 960E4F6C

6E8030E5 364E7229 8C2FF694 C0D41E9A A4D11C98 F4A4266A 912F4BC5
97A0F184 8752910E 25F18C8D 0D2F80C2 B8C9DD17 97FD360B 3938986C
43E6D3A4 70802DEB 39BEDAA8 310A0FCB 4B46F56B

#### F.1.2.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 1742DD26 734FAF9D 9E23C4A9 6D9D7401 712E9682 C45B3D6D 7F305A7E
DFDC6C88

$h(M)$ = BA7816BF 8F01CFEA 414140DE 5DAE2223 B00361A3 96177A9C B410FF61
F20015AD

#### F.1.2.4 Signature

$R$ = 72D12BE8 35E889FC D88AFB32 8C177D78 24BBE734 C0CC35F4 F0A238C3
FCE5873D

$S$ = 8EDABA94 0C0F43E1 D60E9FFC 14A975D5 1DBAEA44 4AD2FD3C 1B1E60FE
47E7938E

#### F.1.2.5 Verification

$\overline{R}$ = 72D12BE8 35E889FC D88AFB32 8C177D78 24BBE734 C0CC35F4 F0A238C3
FCE5873D

## F.2 KCDSA mechanism

A complete explanation of the generation of all values is given in [29], Appendix 6. This example uses the Secure Hash Algorithm (SHA-1) as the hash-function h. The hash-code is simply the value of SHA-1.

### F.2.1 Parameters

$l$ = 200 (i.e., 512 in decimal)

$\alpha$ = 1024

$\beta$ = 160

$P$ = D7B9AFC1 04F4D53F 737DB88D 6BF77E12 CD7EC3D7 1CBE3CB7
4CD224BF F348154A FBA6BFED 797044DF C655DCC2 0C952C0E
C43A97E1 AD67E687 D10729CA F622845D 162AFCA8 F0248CC4
12B3596C 4C5D3384 F7E25EE6 44BA87BB 09B164FB 465477B8
7FDBA5EA A400ffA0 925714AE 19464ffA CEAD3A97 50D12194
8AB2D8D6 5C82379F

$Q$ = C3DDD371 7BF05B8F 8DD725C1 62F0B943 2C6F77FB

$G$ = 50E414C7 A56892D1 AD633E42 D5CD8346 F2C09808 111C772C
C30B0C54 4102C27E 7B5F9BEC 57B9DF2A 15312891 9D795E46
652B2A07 2E1F2517 F2A3AFFF 5815253A AEFE3572 4CFA1AF6
AFCE3A6B 41E3D0E1 3BED0EFF 54383C46 65E69B47 BA79BBC3
339F86B9 BE2B5889 4A18B201 AFC41FE3 A0D93D31 25EFDA79
BC50DBBB 2C3AB639

### F.2.2 Signature key and verification key

$X$ = 068C4EF3 55D8B6F5 3EFF1DF6 F243F985 63896C58

$Y$ = 96DCE0E7 b2F17009 3D9B51D2 BA782027 33B62C40 6D376975
8B3E0CBB A1FF6C78 727A3570 3CB6BC24 76C3C293 743DFEE9
4AA4B9EF A9A17FA6 BF790AC2 5A82C615 23F50ABA AC7B6464
7EB15C95 7B07F5ED 7D467243 089F7469 5CD58FBF 57920CC0
C05D4582 9C0A8161 B943F184 51845760 ED096540 E78AA975
0B03D024 48CDF8DE

### F.2.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 4B037E4B 573BB7E3 34CAD0A7 0BED6B58 81DF9E8E

$Y'$ = 23F50ABA AC7b6464 7Eb15C95 7B07F5ED 7D467243 089F7469
5CD58FBF 57920CC0 C05D4582 9C0A8161 B943F184 51845760
ED096540 E78AA975 0B03D024 48CBF8DE

$h(Y'||M)$ = 915A1B98 E87BAE0E B3D2B480 71423ABD 65124D9D

$V$ = B5C5581A EBD40C45 9EEB0010 36B55D04 AC6D698B

### F.2.4 Signature

$R$ = 249F4382 03AFA24B 2D39B490 47F767B9 C97F2416

$S$ = 15A183ED 488BD8AB 437DB103 163D1B2D DE6DC08D

### F.2.5 Verification

$\overline{R}$ = 249F4382 03AFA24B 2D39B490 47F767B9 C97F2416

## F.3 Pointcheval-Vaudenay mechanism

### F.3.1 Parameters

$L$ = 200 (i.e., 512 in decimal)

$F$ = 2

$P$ = 8DF2A494 492276AA 3D25759B B06869CB EAC0D83A FB8D0CF7
CBB8324F 0D7882E5 D0762FC5 B7210EAF C2E9ADAC 32AB7AAC
49693DFB F83724C2 EC0736EE 31C80291

$Q$ = C773218C 737EC8EE 993B4F2D ED30F48E DACE915F

$G$ = 626D0278 39EA0A13 413163A5 5B4CB500 299D5522 956CEFCB
3BFF10F3 99CE2C2E 71CB9DE5 FA24BABF 58E5B795 21925C9C
C42E9F6F 464B088C C572AF53 E6D78802

### F.3.2 Signature key and verification key

$X$ = 2070B322 3DBA372F DE1C0FFC 7B2E3B49 8B260614

$Y$ = 19131871 D75B1612 A819F29D 78D1B0D7 346F7AA7 7BB62A85
9BFD6C56 75DA9D21 2D3A36EF 1672Ef66 0B8C7C25 5CC0EC74
858FBA33 F44C0669 9630A76B 030EE333

### F.3.3 Per message data

$K$ = 358DAD57 1462710F 50E254CF 1A376B2B DEAADFBF

$K^{-1}$ = 0D516729 8202E49B 4116AC10 4FC3F415 AE52F917

$M$ = ASCII form of "abc" = 61 62 63

### F.3.4 Signature

$R$ = 8BAC1AB6 6410435C B7181F95 B16AB97C 92B341C0

$R||M$ = 8BAC1AB6 6410435C B7181F95 B16AB97C 92B341C0 616263

$h(R||M)$ = 2048680B 36D19516 CF78E869 BEAE7BE9 AB5DC543

$S$ = 5BFDAC3D 665FA38F 6ED315B3 B2F41B86 15187CCD

### F.3.5 Verification

$\overline{\Pi}$ = 2FC6CB9A C3BE0EAC 3DAF02EE FB96FCA3 846708A2 8DD05730
165FE509 42F7F07E DFEF8E52 FCB9369E 3814AA24 607E8047
5D0E61AD 461D6B16 B6CEC5BA AE58946E

$\overline{R}$ = 8BAC1AB6 6410435C B7181F95 B16AB97C 92B341C0

## F.4 EC-DSA mechanism

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.

NOTE – 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.4.1 Example 1: Field $F_2^m$, $m$ =191

#### F.4.1.1 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.4.1.2 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.4.1.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 3EEACE72 B4919D99 1738D521 879F787C B590AFF8 189D2B69

$\Pi$ = ($\Pi_x$, $\Pi_y$).

$\Pi_x$ = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

$\Pi_y$ = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA

$h(M)$ = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

#### F.4.1.4 Signature

$R$ = 038E5A11 FB55E4C6 5471DCD4 998452B1 E02D8AF7 099BB930

$S$ = 0C9A08C3 4468C244 B4E5D6B2 1B3C6836 28074160 20328B6E

#### F.4.1.5 Verification

$\overline{\Pi}$ = ($\overline{\Pi}_x$, $\overline{\Pi}_y$)

$\overline{\Pi}_x$ = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5

$\overline{\Pi}_y$ = 2AD603A0 5BD1D177 649F9167 E6F475b7 E2FF590C 85AF15DA

$\overline{R}$ = 038E5A11 FB55E4C6 5471DCD4 998452B1 E02D8AF7 099BB930

### F.4.2 Example 2: Field $F_P$, 192-bit Prime $P$

#### F.4.2.1 Parameters

The field is $F_P$ where $P$ is in hexadecimal

$P$ = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF

The elliptic curve is: $Y^2 = X^3 + aX + b$ 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.4.2.2 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.4.2.3 Per message data

$M$ = ASCII form of "abc" = 616263

$h(M)$ = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

$K$ = FA6DE297 46BBEB7F 8BB1E761 F85F7DFB 2983169D 82FA2F4E

$\Pi = (\Pi_X, \Pi_Y)$.

$\Pi_X$ = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

$\Pi_Y$ = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835.

#### F.4.2.4 Signature

$R$ = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

$S$ = E9ECC781 06DEF82B F1070CF1 D4D804C3 CB390046 951DF686

#### F.4.2.5 Verification

$\overline{\Pi} = (\overline{\Pi}_X, \overline{\Pi}_Y)$

$\overline{\Pi}_X$ = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

$\overline{\Pi}_Y$ = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835

$\overline{R}$ = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD

## F.5 EC-KCDSA mechanism

A complete explanation of the generation of all values is given in TTAS.KO-12.0015, Appendix VI.

### F.5.1 Example 1: Field $F_2^m$, $m$ =163

This example uses the Secure Hash Algorithm (SHA-1) as the hash-function $h$. The hash-code is simply the value of SHA-1.

#### F.5.1.1 Parameters

The field $F_2^m$ is represented as polynomials modulo the irreducible polynomial $x^{163} + x^8 + x^2 + x + 1$.

The elliptic curve is: $Y^2 + X\,Y = X^3 + a\,X^2 + b$ over $F_2^m$.

$a$ = 7 2546B543 5234A422 E0789675 F432C894 35DE5242

$b$ = 0 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9.

$G = (G_X, G_Y)$

$G_X$ = 7 AF699895 46103D79 329FCC3D 74880F33 BBE803CB

$G_Y$ = 1 EC23211B 5966ADEA 1D3F87F7 EA5848AE F0B7CA9F.

$Q$ = 4 00000000 00000000 0001E60F C8821CC7 4DAEAFC1.

#### F.5.1.2 Signature key and verification key

$X$ = 533FC16B FF07B2FE 15E761BB FB5F0B63 3FD43D42

$Y = (Y_X, Y_Y)$

$Y_X$ = 1 A143BF83 23444956 687C03CA 8DD68CCC 9509DA33

$Y_Y$ = 3 55D548DB 894D5D47 1F813F2E E3E692D1 75B9B6FD

#### F.5.1.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 7D1C44A2 AD7318CF 2C525AAD 16515378 1EB702DA

$\Pi = G^K = (\Pi_x, \Pi_y)$.

$\Pi_x$ = 6 630EE3E5 C71F504B C141A406 AE71E062 441E5E42

$\Pi_y$ = 4 780DDE74 C235C60E C9A25253 DFCC60D4 AA281C6A

$Y'$ = 01A143BF 83234449 56687C03 CA8DD68C CC9509DA 330355D5
48DB894D 5D471F81 3F2EE3E6 92D175B9 B6FD0000 00000000
00000000 00000000 00000000 00000000

$h(Y'||M)$ = 736F9391 4BB28FA7 E93B65D9 C760B5EE 1F8A4305

#### F.5.1.4 Signature

$R$ = A4BFE332 2B286929 3322652E AF349057 F424A6BF

$S$ = 02 633CFF4A 9E81F1E5 266C4A01 7A7C41C0 04034268

#### F.5.1.5 Verification

$\overline{R}$ = A4BFE332 2B286929 3322652E AF349057 F424A6BF

### F.5.2 Example 2: Field $F_P$, 192-bit Prime $P$

This example uses RIPEMD-160 as the hash-function $h$. The hash-code is simply the value of RIPEMD-160.

#### F.5.2.1 Parameters

The field is $\mathbf{F}_P$ where $P$ is

$P$ = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF

The elliptic curve is: $Y^2 = X^3 + aX + b$ 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 631011EC 6B24CDD5 73F977A1 1E794811

$Q$ = FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831

#### F.5.2.2 Signature key and verification key

$X$ = 444811A3 23E03C28 A34CD859 EE2FF1A3 4D1AAF3C B0B5603B

$Y_x = (Y_x, Y_y)$

$Y_x$ = 793C9E6E F7CF74C4 CB8FFB6F 3A2C1A9F E9AEBBB2 8AA7451A

$Y_y$ = B0823C74 7BE23AF0 B170AFB8 13239437 789A03AA 9C526783

#### F.5.2.3 Per message data

$M$ = ASCII form of "abc" = 61 62 63

$K$ = 4B19A072 5424CD33 10B02D8C 8416C98D 64C618BF E935597D

$\Pi = G^K = (\Pi_x, \Pi_y)$.

$\Pi_x$ = DE3DA2DF 1AFC3446 BB5CAA85 0A978D21 19246CDC 0B197E6C

$\Pi_y$ = 15DD6151 225CED60 29D3531F 33092512 89B124EB B15D172B

$Y'$ = 793C9E6E F7CF74C4 CB8FFB6F 3A2C1A9F E9AEBBB2 8AA7451A
B0823C74 7BE23AF0 B170AFB8 13239437 789A03AA 9C526783
00000000 00000000 00000000 00000000

$h(Y'||M)$ = 96BAD51E F1B1CB1F 13710AA7 C0946E0B DFB75BF1

#### F.5.2.4 Signature

$R$ = 3CA29800 D425FCAA 51CCB209 B4ED5D6C 35210822

$S$ = 3143B2EA 5A0E8644 CE8F768A 6FA4D193 C726AD08 019788E5

#### F.5.2.5 Verification

$\overline{R}$ = 3CA29800 D425FCAA 51CCB209 B4ED5D6C 35210822

### F.5.3 Example 2: Field $F_P^m$, 32-bit $P$ and $m = 5$

This example uses RIPEMD-160 as the hash-function $h$. The hash-code is simply the value of RIPEMD-160.

#### F.5.3.1 Parameters

The field is $\mathbf{F}_P^m$ where $P$ is

$P$ = FFFFFFFB

and $m = 5$. The field is represented as polynomials modulo the irreducible polynomial $x^5 - 2$. For convenience, an element of this field is represented as $(a_{m-1} \ \ldots \ a_1 \ a_0)$.

The elliptic curve is: $Y^2 = X^3 + aX + b$ over $F_P^m$.

$a$ = (00000000 00000000 00000000 00000000 00000001)

$b$ = (00000000 00000000 00000000 00000001 00000106)

$G = (G_x, G_y)$

$G_x$ = (FCDEE3EE EB6A9D0C 821C8B46 D27937BC 0FBAC840)

$G_y$ = (3C329E0D 7A5FB6E4 048A69C1 12F8CB35 DFFB7CCC)

$Q$ = FFFFFFE7 000000F9 FFFE3308 F697C1D6 D7DE35CF

#### F.5.3.2 Signature key and verification key

$X$ = CA7DCCDA 50098667 936876B4 72EDFCFC 8F613136

$Y_x = (Y_x, Y_y)$

$Y_x$ = (2A51DC89 0265DCCC 6DD5715F 2DA674AF 8AE1CF75)

$Y_y$ = (F488AED5 DDFC9BCA B4B9EDB3 FC3F93F2 0660A53D)

#### F.5.3.3 Per message data

$M$ = ASCII form of “abc” = 61 62 63

$K$ = 792D8EC4 2B8377EF BCD9BFAB 00348E4E 63F510F7

$\Pi = G^K = (\Pi_x, \Pi_y)$.