INTERNATIONAL STANDARD

ISO/IEC 14888-3

First edition
1998-12-15

Corrected and reprinted
1999-12-15

Information technology — Security techniques — Digital signatures with appendix —

Part 3:
Certificate-based mechanisms

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

Partie 3: Mécanismes fondés sur certificat

Reference number
ISO/IEC 14888-3:1998(E)

Foreword

ISO (the International Organization for Standardization) and the 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. 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.

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

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: Identity-based mechanisms*

— *Part 3: Certificate-based mechanisms*

Further parts may follow.

Annexes A and B form an integral part of this part of ISO/IEC 14888. Annexes C to G are for information only.

Information technology — Security techniques — Digital signatures with appendix —

Part 3:
Certificate-based mechanisms

1 Scope

ISO/IEC 14888 specifies digital signature mechanisms with appendix for messages of arbitrary length and is applicable for providing data origin authentication, non-repudiation, and integrity of data.

This part of ISO/IEC 14888 specifies certificate-based digital signature mechanisms with appendix. In particular, this part of ISO/IEC 14888 provides 1) a general description of certificate-based digital signature mechanisms whose security is based on the difficulty of the discrete logarithm problem in the underlying commutative group (see Clause 6), 2) a general description of certificate-based digital signature mechanisms whose security is based on the difficulty of factoring (see Clause 7), and 3) a variety of normative digital signature mechanisms with appendix using certificate-based mechanisms for messages of arbitrary length (see Annex A and B).

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 14888. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 14888 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 14888-1:1998, *Information technology — Security techniques — Digital signatures with appendix — Part 1: General*.

ISO/IEC 14888-2:1999, *Information technology — Security techniques — Digital signatures with appendix — Part 2: Identity-based mechanisms*.

ISO/IEC 9796:1991, *Information technology — Security techniques — Digital signature scheme giving message recovery*.

ISO/IEC 9796-2:1997, *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Mechanisms using a hash-function*.

ISO/IEC 10118-3:1998, *Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*.

ISO/IEC 10118-4:1998, *Information technology — Security techniques — Hash-functions — Part 4: Hash-functions using modular arithmetic*.

3 General

This part of ISO/IEC 14888 makes use of the definitions, symbols, legend for figures, and notation given in ISO/IEC 14888-1.

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. For certificate-based mechanisms, this association must be provided by some certifying measure, for example, the verification key is retrieved from a certificate.

The goal of this part of ISO/IEC 14888 is to specify the following processes and functions within the general model described in ISO/IEC 14888-1:

- the process of generating keys
    - generating domain parameters
    - generating signature and verification keys

- the process of producing signatures
    - (optional) producing pre-signatures
    - preparing the message for signature

- computing witnesses
- computing the signature

- the process of verification
    - preparing message for verification
    - retrieving the witness
    - computing the verification function
    - verifying the witness

4 Definitions

For the purpose of this part of ISO/IEC 14888, the definitions of ISO/IEC 14888-1 apply. Additional definitions which are required are as follows.

4.1 Finite commutative group: A finite set J with the binary operation «∗« such that:

- For all a, b, c∈J, (a∗b) ∗ c = a ∗ (b∗c)
- There exists e∈J with e∗a = a for all a∈J
- For all a∈J there exists b∈J with b∗a = e
- For all a, b∈J, a∗b = b∗a

4.2 Order of an element in a finite commutative group: If a^0 =e, and a^{n+1}=a∗a^n (for n ≥ 0), is defined recursively, the order of a∈J is the least positive integer n such that a^n = e.

5 Symbols and notation

Throughout this part of ISO/IEC 14888 the following symbols and notations are used in addition to those given in ISO/IEC 14888-1.

$\boldsymbol{E}$	a finite commutative group
$\#\boldsymbol{E}$	the cardinality of $\boldsymbol{E}$
a||b	concatenation of b to a
Q	a divisor of $\#\boldsymbol{E}$
G	an element of order Q in $\boldsymbol{E}$
gcd(U, N)	the greatest common divisor of integers U and N
T_1	first part of assignment
T_2	second part of assignment
$\boldsymbol{Z}_N$	the set of integers U with $0 \le U < N$
$\boldsymbol{Z}_N^*$	the set of integers U with $0 < U < N$ and gcd (U, N) = 1
$\lfloor a \rfloor$	the greatest integer equal to or less than a

6 Digital signature mechanisms based on discrete logarithms

6.1 Key generation process

6.1.1 Generating domain parameters

For digital signature mechanisms based on discrete logarithms, the set $\boldsymbol{Z}$ of domain parameters determines the following parameters:

- a finite commutative group $\boldsymbol{E}$
- one or more divisors Q of $\#\boldsymbol{E}$
- one or more elements G of order Q in $\boldsymbol{E}$

In the group $\boldsymbol{E}$, multiplicative notation is used. The signature mechanism will use one element G in $\boldsymbol{E}$. It is worthwhile to note that the particular signature mechanism chosen may place additional constraints on the choice of $\boldsymbol{E}$, Q, and G.

6.1.2 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 $0 < X < Q$ and gcd(X, Q) = 1. The corresponding public verification key Y is an element of $\boldsymbol{E}$ and is computed as

$$Y = G^X .$$

Note: It is allowed to exclude a few integers from consideration as possible X values.

In some instances, validation of domain parameters and keys may be required. However, it is outside the scope of this standard.

6.2 Signature process

In this clause the signature process for a class of signature mechanisms is described. Within this class the signature function for the mechanism to be used is specified by a permutation (A, B, C) of (S, T_1, T_2) which determines the coefficients of the signature equation.

$$\text{A}K + \text{B}X + \text{C} \equiv 0 \pmod{Q}.$$

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

The signature process and the formation of a signed message consists of eight stages (See Figure 1):

- producing the randomizer
- producing the pre-signature
- preparing the message for signing

- computing the witness (the first part of the signature)
- computing the assignment
- 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 X, and the domain parameters $\mathbf{E}$, G, and Q.

6.2.1 Producing the randomizer

The signing entity generates a secret randomizer which is an integer K with $0 < K < Q$ and satisfying gcd $(K, Q) = 1$. The output of this stage is K, which the signing entity keeps secret.

Note: It is allowable to exclude a few integers from consideration as possible K values.

6.2.2 Producing the pre-signature

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

$$\Pi = G^K$$

in $\mathbf{E}$. The output of this stage is the pre-signature, Π.

6.2.3 Preparing the message for signing

The message is split into two parts which will be called data inputs M_1 and M_2. One of these parts may be empty and the two parts need not be distinct (See ISO/IEC 14888-1 for further details.)

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

The variables to this stage are the pre-signature Π from 6.2.2 and M_1 from 6.2.3. The values of these variables are taken as inputs to the witness function. The output of the witness function is witness R.

6.2.5 Computing the assignment

The inputs to the assignment function are the first part of the signature, which is the witness R from 6.2.4, and M_2 from 6.2.3. The output of the assignment function is assignment $T = (T_1, T_2)$ where T_1 and T_2 are integers such that

$$0 < |T_1| < Q, \ 0 < |T_2| < Q.$$

6.2.6 Computing the second part of the signature

The inputs to this stage are randomizer K from 6.2.1, the signature key X, assignment $T = (T_1, T_2)$ from 6.2.5, the permutation (A, B, C) of (S, T_1, T_2) and domain parameter Q as specified in 6.1.1. The signing entity forms the signature equation

$$(\mathrm{A}K + \mathrm{B}X + \mathrm{C}) \equiv 0 \pmod{Q}$$

and solves the signature equation for S, the second part of the signature, where $0<S<Q$. The pair (R, S) will be called the signature, Σ.

6.2.7 Constructing the appendix

The appendix is constructed from the signature and an optional text field, *text*, as $((R, S), \textit{text})$. The text field could include a certificate which 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.

6.2.8 Constructing the signed message

The signed message is obtained by the concatenation of message M and appendix,

$$M \,||\, ((R, S), \textit{text})$$

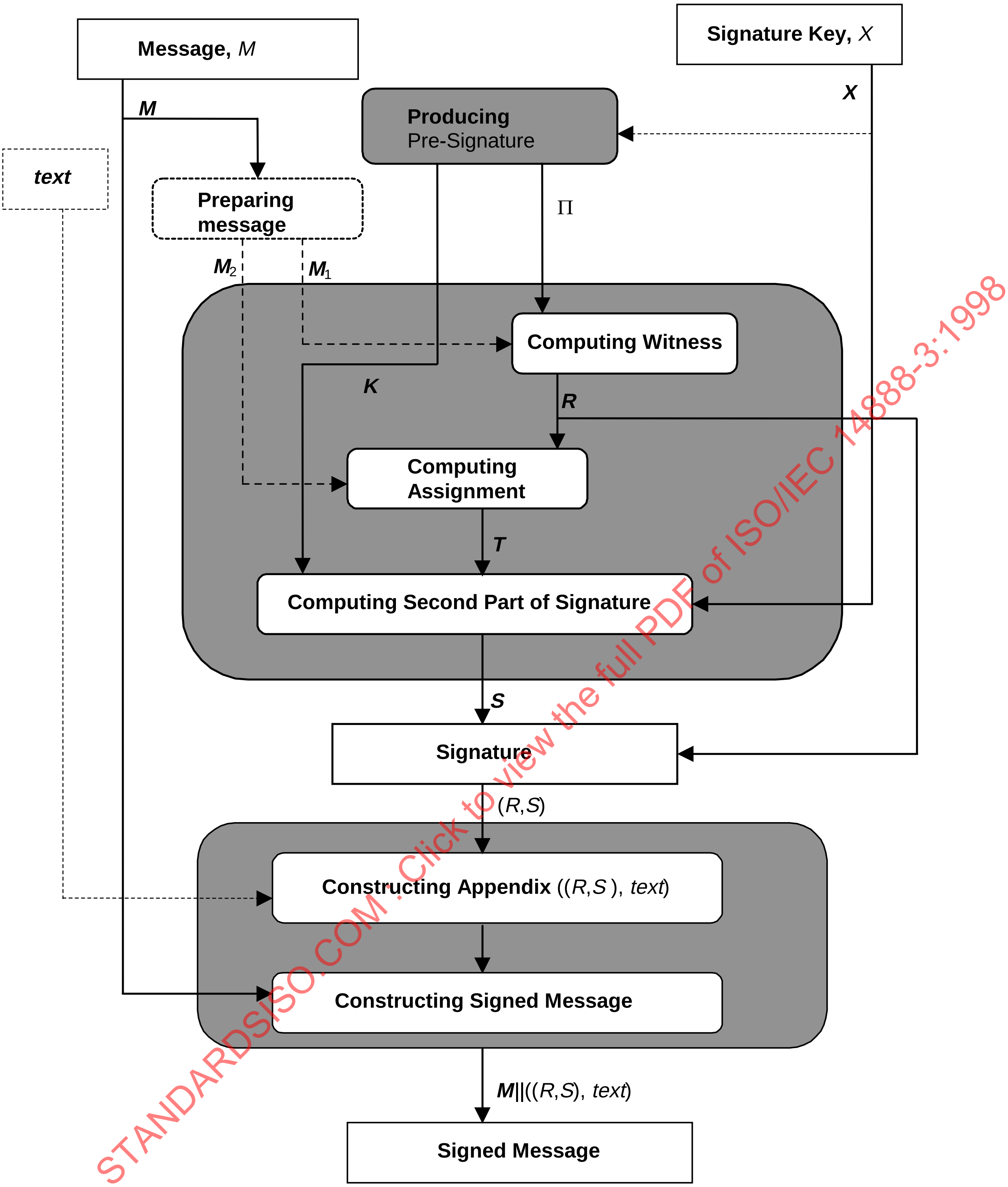

Figure 1 — Signature process with randomized witness

6.3 Verification process

The verification process consists of four stages (See Figure 2).

- Preparing message for verification
- Retrieving the witness
- Computing the verification function
    - retrieving the assignment
    - recomputing the pre-signature
    - recomputing the witness
- Verifying the witness.

In this process, the verifier makes use of the signer's verification key *Y* and the domain parameters: finite group ***E***, element *G* in ***E*** and its order *Q*.

6.3.1 Preparing message for verification

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

6.3.2 Retrieving the witness

The verifier retrieves the signature (*R*, *S*) from the appendix, and divides it into witness *R* and the second part of the signature *S*.

6.3.3 Computing the verification function

6.3.3.1 Retrieving the assignment

This stage is identical to 6.2.5. The inputs to the assignment function consist of the witness *R* from 6.3.2 and M_2 from 6.3.1. The assignment $T = (T_1, T_2)$ is recomputed as the output from the assignment function.

6.3.3.2 Recomputing the pre-signature

The inputs to this stage are the set ***Z*** of domain parameters, the verification key *Y*, the assignment $T = (T_1, T_2)$ from 6.3.3.1 and the second part of the signature *S* from 6.3.2. The verifier assigns to the coefficients (A, B, C) the values (S, T_1, T_2) according to the order specified by the signature function, and computes the element $\overline{\Pi}$ in ***E*** as

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

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

6.3.3.3 Recomputing the witness

The computations at this stage are the same as in 6.2.4. The verifier executes the witness function. The inputs are $\overline{\Pi}$ from 6.3.3.2 and M_1 from 6.3.1. The output is the recomputed witness, $\overline{R}$.

6.3.4 Verifying the witness

The signature is verified if the recomputed witness, $\overline{R}$ from 6.3.3.3 is equal to *R* from 6.3.2. Additional checks may be required (See A.1.2.4.6 for other example checks.)

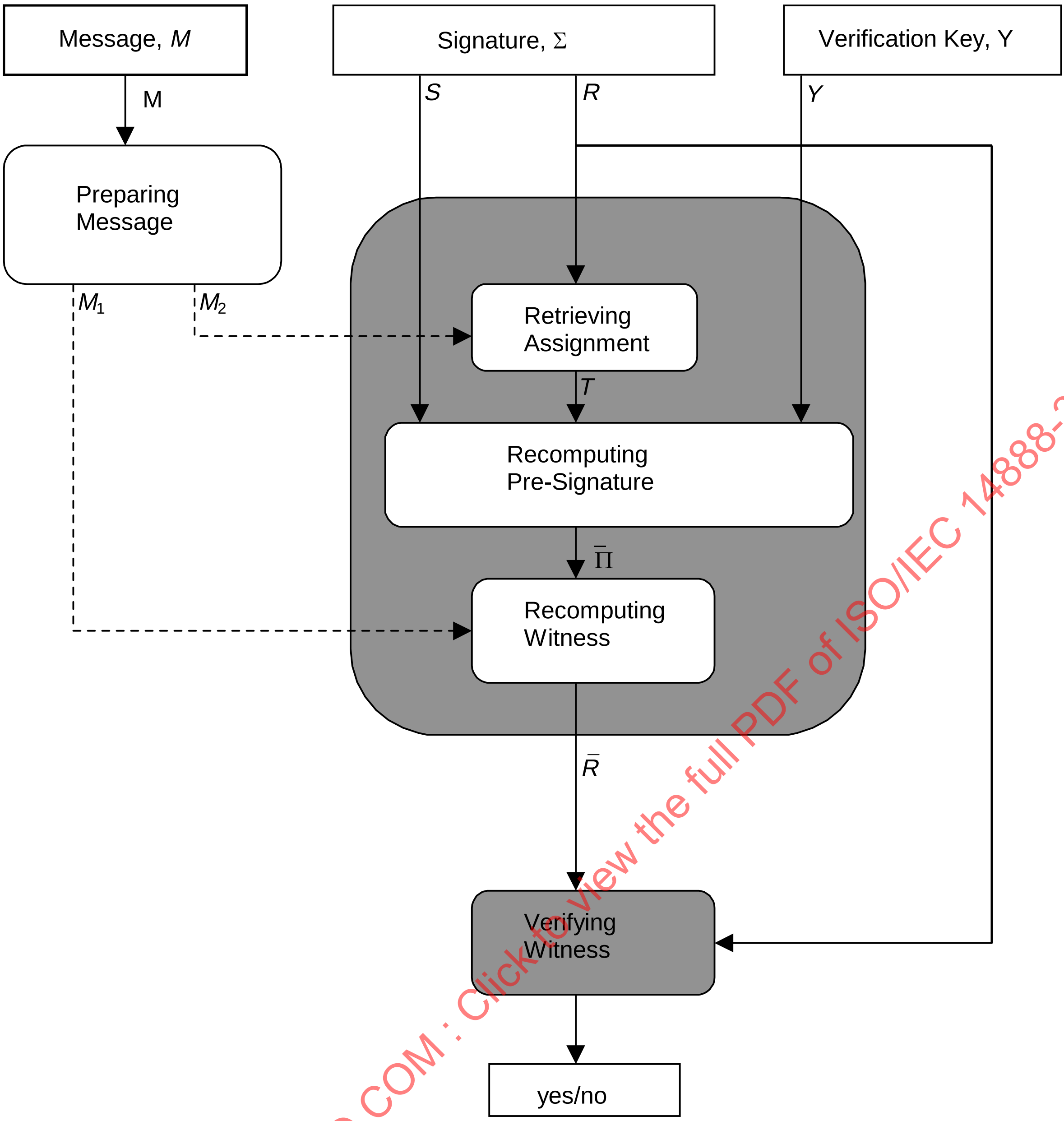

Figure 2 — Verification process with a randomized witness

7 Digital signature mechanisms based on factoring

Digital signature mechanisms based on factoring utilize a deterministic witness and produce a one-part signature, but can be randomized or deterministic (Reference ISO/IEC 14888-1, Figures 2 and 4). In either case, such a mechanism employs an integer *N* as a component of the verification key whose factorization is part of the signature key. It is assumed that it is computationally infeasible to factor *N* into its prime factors. Constraints should be imposed on the generation of the signature key to make the factorization sufficiently difficult.

7.1 Key generation process

7.1.1 Generation of domain parameters

For digital signature mechanisms based on factoring, the set ***Z*** of domain parameters optionally contains an integer *v* used as a system wide portion of the verification key, subject to the conditions specified in 7.1.2.

7.1.2 Generation of signature key and verification key

7.1.2.1 Generation of signature key

A signature key of a signing entity is a secretly generated collection $X = (\{P_1, P_2, \ldots, P_r\}, s)$, consisting of a set of randomly or pseudo-randomly chosen, but not necessarily distinct prime integers P_i, and an integer *s*. The minimum number of distinct primes to be used is two.

7.1.2.2 Generation of verification key

The verification key *Y* is a pair of integers (*N*, *v*) where *N* is the product, $\Pi\ P_i$ of all primes P_i and *v* is an integer which satisfies a condition depending on the signature key.
If *v* is specified as a domain parameter, additional constraints might be imposed on the signature key so that *v* satisfies the appropriate condition.

7.2 Signature process

7.2.1 Producing the pre-signature (optional)

A randomized signature mechanism employs a pre-signature, which depends only on a randomizer and a signature key. The pre-signature is computed in two steps.

7.2.1.1 Producing the randomizer

The signing entity secretly generates a randomizer which is an integer *K* mod *N*, possibly subject to additional constraints. The output of this stage is *K*, which the signing entity keeps secret.

7.2.1.2 Computing the pre-signature

The pre-signature is a function of the randomizer and independent of the message. The input to this stage is the randomizer *K* and the signature key. The output of this stage is the pre-signature, denoted Π.

7.2.2 Preparing of message for signing

The message is used to construct data inputs M_1 and M_2. The second part, M_2, might be empty and the two inputs need not be distinct.

7.2.3 Computing the witness

The input to this stage is the data input M_1. The output is the hash token, *H*, determined by the data input M_1. Note that the hash token is interpreted as an integer mod *N* chosen so that $0 < H < N$.

7.2.4 Computing the signature

The inputs to this stage are the witness computed in 7.2.3, the signature key from 7.1.2.1 and optional data input M_2 (See ISO/IEC 14888-1, Figure 2). For a randomized mechanism, the randomizer *K* and the pre-signature Π are also valid inputs. The output is a one-part signature $\Sigma = S$.

7.2.5 Constructing the appendix

The appendix is constructed from the signature, Σ and an optional text field, *text*. The text field could include a certificate which cryptographically ties the public verification key to the identification data of the signing entity.

7.2.6 Constructing the signed message

The signed message is obtained by concatenating the message *M* with the appendix from 7.2.5,

$$M \,||\, (\Sigma\ ,\ text\ ).$$

7.3 Verification process

7.3.1 Preparing message for verification

The verifier retrieves *M* from the signed message and determines the two data input parts M_1 and M_2 as specified in 7.2.2.

7.3.2 Retrieving the witness

The verifier retrieves the value of the witness *H* as a function of the data input M_1 according to the witness function specified in 7.2.3.

7.3.3 Computing the verification function

Using the integer *v* obtained either from the domain parameter set ***Z*** or the verification key *Y*, the verifier uses the verification function to obtain a recomputed witness, $\overline{H}$.

7.3.4 Verifying the witness

The signature is valid if the value of the retrieved witness H agrees with the value from the verification function of the recomputed witness, $\overline{H}$.

Annex A
(normative)

Examples of certificate-based digital signatures with appendix based on discrete logarithms

Examples of such signature mechanisms are the Digital Signature Algorithm (DSA) of the U.S. NIST, Pointcheval/Vaudenay, and elliptic curve signatures. These schemes are described below.

The groups used for the signature mechanisms include a multiplicative group $\mathbf{Z}^*_P$, where P is a prime (i.e., DSA and Pointcheval/Vaudenay) and an additive group formed by the points of an elliptic curve over a finite field (i.e., Elliptic Curve DSA).

A.1 Non-Elliptic curve based examples

A.1.0 Symbols and notation

P	prime integer
$\boldsymbol{Z}_P$	set of integers U with $0 \le U < P$
$\boldsymbol{Z}^*_P$	set of integers U with $0 < U < P$

A.1.1 The U.S. Digital Signature Algorithm (DSA)

This example is taken from the U.S. National Institute of Standards and Technology (NIST) Federal Information Processing Standards Publication 186 (FIPS PUB 186), 19 May 1994. The general parameters defined in clause 6 shall have the following forms. The notation here has been changed slightly from FIPS PUB 186 to conform with notation used elsewhere in this part of ISO/IEC 14888.

The DSA is a signature mechanism with $\boldsymbol{E} = \boldsymbol{Z}^*_P$, P a prime, and Q a prime dividing P - 1. The message is split such that M_1 is empty and $M_2 = M$. 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-token of message M, converted to an integer according to the conversion rule given in Annex C. The hash-function h is the Secure Hash Algorithm (SHA) as adopted in the U.S. NIST Secure Hash Standard (SHS), FIPS PUB 180-1, 17 April 1995. The Secure Hash Algorithm is also described in ISO/IEC DIS 10118-3. (Note that no control field with a hash-function identifier is required for DSA, thus the hash token is simply h(M). See ISO/IEC 14888-1).

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

$$(SK - RX - H) \equiv 0 \pmod{Q}.$$

A.1.1.1 DSA Parameters

L	512 + 64l, for l an integer $0 \le l < 8$
P	a prime, where $2^{L-1} < P < 2^L$
Q	a prime divisor of P-1, where $2^{159} < Q < 2^{160}$
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 $\boldsymbol{E} = \boldsymbol{Z}^*_P$

The integers P, Q, and G can be public and can be common to a group of users.
To achieve FIPS compliance, parameters P and Q are generated as specified in FIPS PUB 186, Appendix 2 (Details can be found in Annex C of this part of ISO/IEC 14888).

Note 1: The size of the prime P in this normative example is as specified by the Digital Signature Algorithm (DSA). Note that the size of P is restricted to be at most 1024 bits. As of 19 May 1994, the size of P provides a sufficient security margin. It is acknowledged that future advances in number theoretic algorithms may possibly render the size of P of 1024 bits as insufficient.

Note 2: It is recommended that all users check the proper generation of the DSA public parameters.

Note 3: It is recognized that DSA possesses an unfavourable property in which an attack can be mounted where collisions on the underlying hash function can be found with a complexity of 2^{74} as compared to 2^{80} in the most secure case. This attack though is easily detectable. For users who may still wish to avoid this property, it can be prevented by using the mechanism of A.1.2.

A.1.1.2 DSA 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 corresponding public verification key Y is

$$Y = G^X .$$

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

A.1.1.3 DSA signature process

A.1.1.3.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0<K<Q$. Parameter K must be generated for each signature and must be kept secret.

A.1.1.3.2 Producing the pre-signature

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

$$\Pi = G^K \bmod P$$

A.1.1.3.3 Preparing the message for signing

The message is split such that M_1 is empty and M_2 is the message, $M_2 = M$.

A.1.1.3.4 Computing the witness

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

$$R = (G^K \bmod P) \bmod Q$$

A.1.1.3.5 Computing the assignment

The signing entity computes the assignment $(T_1,T_2) = (-R,-H)$ where $H = h(M)$ is the hash-token of message M and $M = M_2$.

A.1.1.3.6 Computing the second part of the signature

The signature is (R, S). Thus,

$$R = (G^K \bmod P) \bmod Q$$
$$S = (K^{-1}(h(M) + XR)) \bmod Q$$

The value of $h(M)$ is a 160-bit string output of the Secure Hash Algorithm. For use in computing S, this string must be converted to an integer. The conversion rule is given in Annex C.

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).

A.1.1.3.7 Constructing the appendix

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

A.1.1.3.8 Constructing the signed message

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

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

A.1.1.4 DSA 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 (see ISO/IEC 14888-1, clause 9 for additional required data items).

A.1.1.4.1 Preparing the message for verification

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

A.1.1.4.2 Retrieving the witness

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

A.1.1.4.3 Retrieving the assignment

This stage is identical to A.1.1.3.5. The inputs to the assignment function consist of the witness R from A.1.1.4.2 and M_2 from A.1.1.4.1. The assignment $T = (T_1,T_2)$ is recomputed as output from the assignment function, A.1.1.3.5.

A.1.1.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1,T_2)$ from A.1.1.4.3 and second part of the signature S from A.1.1.4.2. The verifier assigns the coefficients (A, B, C) the values (S,T_1,T_2) as determined by the signature function, and obtains a recomputed value $\overline{\Pi}$ of the pre-signature using the formula

$$\overline{\Pi} = Y^{-A^{-1}B \bmod Q} G^{-A^{-1}C \bmod Q} \bmod P \text{ in } \mathbf{E}.$$

A.1.1.4.5 Recomputing the witness

The computations at this stage are the same as in A.1.1.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from A.1.1.4.4. Note that

M_1 is empty. The output is the recomputed witness $\overline{R}$.

A.1.1.4.6 Verifying the witness

Let M_2 be the value from A.1.1.4.1, and R and S the values from A.1.1.4.2. Let Y be the public verification key of the signing entity. To verify the signature, the verifier first checks to see that $0 < R < Q$ and $0 < S < Q$. If either condition is violated the signature shall be rejected. If these two conditions are satisfied, the verifier compares the recomputed witness, $\overline{R}$ from A.1.1.4.5 to the value of R from A.1.1.4.2. If $\overline{R} = R$, then the signature is valid.

A.1.2 Pointcheval/Vaudenay signatures

The method of Pointcheval/Vaudenay is a variant of the DSA algorithm, with $\boldsymbol{E} = Z^*_P$, P a prime, and Q a prime divisor of P-1. The message is split such that M_1 is empty and $M_2 = M$. The witness is defined by the formula

$$R = \Pi \bmod Q$$

and the assignment function by the formula

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

where $H = h(R\,||M)$ is the hash token of the concatenation of the witness R and the message M. The hash-function h is the Secure Hash Algorithm (SHA-1). Note that the computation of T_2 above requires the conversion of the hash code to an integer. Some agreed upon method for this conversion is required for this step (see for example ISO/IEC DIS 10118-4) .

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

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

A.1.2.1 Pointcheval/Vaudenay parameters

P	prime number
Q	prime divisor of P-1
F	integer such that $1 < F < P$-1 and $F^{(P-1)/Q} \bmod P > 1$
G	$F^{(P-1)/Q} \bmod P$

Note: Special care should be taken to the generation of P, Q, and F. For example, the procedures of A.1.1.1 may be used.

A.1.2.2 Pointcheval/Vaudenay 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 corresponding public verification key Y is

$$Y = G^X.$$

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

A.1.2.3 Pointcheval/Vaudenay signature process

A.1.2.3.1 Producing the randomizer

The signing entity computes a random or pseudo-random integer K such that $0<K<Q$ and $\gcd(K,Q) = 1$.

A.1.2.3.2 Producing the pre-signature

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

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

A.1.2.3.3 Preparing message for signing

The message is split such that M_1 is empty and M_2 is the message, $M_2 = M$.

A.1.2.3.4 Computing the witness

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

$$R = (G^K \bmod P) \bmod Q$$

A.1.2.3.5 Computing the assignment

The signing entity computes the assignment $(T_1, T_2) = (-R, -H)$, where $H = h(R||M)$ is the hash token of the concatenation of the witness and message M (and $M = M_2$).

A.1.2.3.6 Computing the signature

The signature is (R, S). Thus,

$$R = (G^K \bmod P) \bmod Q$$
$$S = K^{-1}(h(R||M) + XR) \bmod Q.$$

A.1.2.3.7 Constructing the appendix

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

A.1.2.3.8 Constructing the signed message

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

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

A.1.2.4 Pointcheval/Vaudenay 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 and the other necessary data items.

A.1.2.4.1 Preparing the message for verification

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

A.1.2.4.2 Retrieving the witness

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

A.1.2.4.3 Retrieving the assignment

This stage is identical to A.1.2.3.5. The inputs to the assignment function consist of the witness R from A.1.2.4.2 and M_2 from A.1.2.4.1. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, A.1.2.3.5.

A.1.2.4.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key Y, assignment $T = (T_1, T_2)$ from A.1.2.4.3 and second part of the signature S from A.1.2.4.2. The verifier assigns the coefficients (A, B, C) the values (S, T_1, T_2) as determined by the signature function, and obtains a recomputed value $\overline{\Pi}$ of the pre-signature by computing it using the formula

$$\overline{\Pi} = Y^{-A^{-1}B \bmod Q}\, G^{-A^{-1}C \bmod Q} \bmod P$$

in ***E***.

A.1.2.4.5 Recomputing the witness

The computations at this stage are the same as in A.1.2.3.4. The verifier executes the witness function. The inputs are $\overline{\Pi}$ from A.1.2.4.4 and M_1 from A.1.2.4.1. The output is the recomputed witness $\overline{R}$.

A.1.2.4.6 Verifying the witness

Let M_2 be the value from A.1.2.4.1, and R and S the values from A.1.2.4.2. The verifier checks to see that $0 < R < Q$ and $0 < S < Q$. If either condition is violated the signature shall be rejected.

If these two conditions are satisfied, the verifier compares the recomputed witness, $\overline{R}$ from A.1.2.4.5 to the value of R from A.1.2.4.2. If $\overline{R} = R$, then the signature is valid.

A.2 Elliptic curve based example

A.2.1 Elliptic curve DSA

The following scheme is an elliptic curve analogue of the DSA algorithm. [See Annex D for additional elliptic curve mathematical background information.] Thus it is a signature mechanism with ***E*** being a cyclic group of points on an elliptic curve. We take

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

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

Thus the signature equation becomes

$SK - RX + H \equiv 0 \pmod Q$.

A.2.1.1 Elliptic curve DSA parameters

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

Note: Although it is standard in the literature to write the arithmetic of elliptic curve groups additively, we will be consistent with the general description above and use multiplicative notation.

A.2.1.2 Elliptic curve DSA 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 corresponding public verification key Y is

$Y = G^X$.

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

A.2.1.3 Elliptic curve DSA signature process

A.2.1.3.1 Producing the randomizer

The random secret integer K is generated, $0 < K < Q$.

A.2.1.3.2 Producing the pre-signature

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

$\Pi = G^K$.

A.2.1.3.3 Preparing message for signing

The message is split such that M_1 is empty and M_2 is the message, $M_2 = M$.

A.2.1.3.4 Computing the witness

The signing entity computes $R = \Pi_x \bmod Q$ where Π_x is the x coordinate of the point Π, interpreted as an integer in the range [1, Q-1] (See ISO/IEC 14888-1, subclause 5.2).

A.2.1.3.5 Computing the assignment

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

A.2.1.3.6 Computing the second part of the signature

The signature is (R, S). Thus,

$$R = \Pi_x \bmod Q$$
$$S = (K^{-1}(XR - H)) \bmod Q$$

So that

$$(R, S) = ((\Pi_x) \bmod Q, (K^{-1}(XR - H)) \bmod Q)$$

A.2.1.3.7 Constructing the appendix

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

A.2.1.3.8 Constructing the signed message

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

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

A.2.1.4 Elliptic curve DSA verification process

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

A.2.1.4.1 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 $M_2 = M$.

A.2.1.4.2 Retrieving the witness

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

A.2.1.4.3 Retrieving the assignment

This stage is identical to A.2.1.3.5. The inputs to the assignment function consist of the witness R from A.2.1.4.2 and M_2 from A.2.1.4.1. The assignment $T = (T_1, T_2)$ is recomputed as output from the assignment function, A.2.1.3.5.

A.2.1.4.4 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1, T_2)$ from A.2.1.4.3 and second part of the signature S from A.2.1.4.2. The verifier assigns the coefficients (A, B, C) the values (S, T_1, T_2) as determined by the signature function, and obtains a recomputed value $\overline{\Pi}$ of the pre-signature by computing it using the formula

$$\overline{\Pi} = G^{-A^{-1}C \bmod Q}\, Y^{A^{-1}B \bmod Q}$$

A.2.1.4.5 Recomputing the witness

The computations at this stage are the same as in A.2.1.3.4. The verifier executes the witness function. The input is $\overline{\Pi}$ from A.2.1.4.4. The output is the recomputed witness $\overline{R}$.

A.2.1.4.6 Verifying the witness

Let M, R, and S be the values retrieved from the signed message, and let Y be the public verification key of the signer. To verify the signature, the verifier first checks to see that $0 < R < Q$ and $0 < S < Q$; if either condition is violated the signature shall be rejected. If these two conditions are satisfied, the verifier compares the recomputed witness, $\overline{R}$ from A.2.1.4.5 to the retrieved version of R from A.2.1.4.2.

If $\overline{R} = R$, then the signature is verified.

Annex B
(normative)

Example of certificate-based digital signatures with appendix based on factoring

Examples of such signature mechanisms are digital signatures with hashing based on ISO/IEC 9796 (deterministic) and ESIGN (randomized). These schemes are described below.

B.1 Digital signatures with hashing based on ISO/IEC 9796

The digital signature mechanism given in ISO/IEC 9796 is a deterministic signature mechanism based on factoring. As such, it does not employ a randomizer or pre-signature. There are exactly two secret prime factors P_1, P_2 in the signature key defined in clause 7.

B.1.1 Generation of the domain parameters

The domain parameters ***Z*** optionally contain a specification for a system wide verification exponent v. Other system parameters such as a hash function are optionally specified in the domain parameters.

B.1.2 Generation of the signature key and verification key

B.1.2.1 Public Verification Exponent

If not specified in the domain parameter set, the signing entity selects a positive integer v, where $v < N$ (modulus).

B.1.2.2 Generation of signature key

The signing entity secretly generates a collection $\{P_1, P_2\}$ of two randomly or pseudo-randomly chosen and distinct prime integers P_i, subject to the following conditions

- if v is odd, then P_i-1 shall be coprime to v

- if v is even, then $(P_i - 1)/2$ shall be coprime to v and $P_1 - P_2$ shall not be divisible by 8.

Additional constraints on the P_i to ensure that the factorization of $N = P_1P_2$ is computationally infeasible are optional.

The signing entity computes public modulus $N = P_1P_2$ and the signature exponent, s, an integer mod N with $0 < s < N$ so that

$$sv \equiv 1 \mod \mathrm{lcm}(P_1 - 1, P_2 - 1) \text{ if } v \text{ is odd,}$$

$$sv \equiv 1 \mod \tfrac{1}{2}\, \mathrm{lcm}\,(P_1 - 1, P_2 - 1) \text{ if } v \text{ is even.}$$

The signature key X is the set $(\{P_1, P_2\}, s)$.

B.1.2.3 Generation of verification key

The verification key Y is the set (N, v).

B.1.3 Signature process

The signature process is that of a deterministic signature mechanism, and as such does not produce a pre-signature.

B.1.3.1 Preparing the message for signing

The data input $M_1 = M$ is the message; M_2 is empty.

B.1.3.2 Computing the witness

The deterministic witness is an integer H Mod N, determined by the hash token of the message. The hash token is formed from a padded hash code as defined in ISO/IEC 10118 concatenated with an optional control field containing hash function identification. If the hash function is not uniquely specified by the domain parameter, the control field is mandatory. If the verification key is even, the resulting hash token is forced to have Jacobi symbol 1 mod N by dividing by 2 if necessary.

B.1.3.3 Computing the signature

The signature is $S = H^s \mod N$.

B.1.3.4 Constructing the appendix

The appendix is constructed from the signature and an optional text field, *text*. The text field could include a certificate which cryptographically ties the public verification key to the identification data of the signing entity.

B.1.3.5 Constructing the signed message

The signed message is obtained by the concatenation of message M and the appendix,

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

B.1.4 Verification process

The verifying entity acquires the necessary data items required for the verification process (see ISO/IEC 14888-1, clause 9).

B.1.4.1 Preparing the message for verification

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

B.1.4.2 Retrieving the witness

The witness H is reconstructed from the data input M_1 according to B.1.3.2.

B.1.4.3 Computing the verification function

Using the integer v obtained either from the domain parameters ***Z*** or the verification key Y, and the integer N from the verification key Y, the verifier computes

$$\overline{H} = S^v \bmod N.$$

If the verification exponent is even, $\overline{H}$ is modified according to its congruence modulo 8.

B.1.4.4 Verifying the witness

The signature is valid only if the value of the retrieved witness H agrees with the value of the recomputed witness $\overline{H}$.

B.2 ESIGN

B.2.1 Generation of domain parameters

ESIGN is a digital signature mechanism which uses as a modulus an integer $N = P^2Q$ where $P>Q$ are prime integers and a signature exponent s equal to the verification exponent v, an integer greater than or equal to 4. This common exponent can be included in the domain parameters or derived from a certificate in the optional text of the appendix. Also specified (optionally) in the domain parameters is an integer n which specifies the size of the integer primes in bits. Nominally, n is 1/3 the number of bits used to represent N. The size of the hash token is restricted to n-1 bits (i.e., $0 < H < 2^{n-1}$).

B.2.2 Generation of signature key and verification key

B.2.2.1 Generation of signature key

The signature key of a signing entity is a secretly generated collection $X = (\{P_1, P_2, P_3\}, s)$, determined by two distinct randomly or pseudo-randomly chosen prime integers $P_1 = P_2 = P$ and $P_3 = Q$ with $P>Q$ and the signature exponent s with $s \geq 4$. The factors P and Q shall be kept secret.

B.2.2.2 Generation of verification key

The verification key is a pair of integers $Y = (N, v)$, where N is the product $N = P_1P_2P_3 = P^2Q$ and v is an integer which satisfies the condition $v = s \geq 4$.

B.2.3 Signature process

The signature process of ESIGN follows the general model described in Clause 8 of ISO/IEC 14888-1. It is a randomized signature mechanism which uses a deterministic witness and produces a one-part signature.

B.2.3.1 Producing pre-signature

The pre-signature is computed in two steps.

B.2.3.1.1 Producing the randomizer

The signing entity generates secretly a randomizer which is a random or pseudo-random positive integer K Mod PQ such that $0 < K < PQ$. The output of this stage is K, which the signing entity keeps secret.

B.2.3.1.2 Producing the pre-signature

The input to this stage is the randomizer K and the signature key X. The signing entity computes the pre-signature $\Pi = (U, V)$, where $U = K^s \bmod N$ and $V = (sK^{s-1})^{-1} \bmod P$. The second part V of the pre-signature shall be kept secret.

B.2.3.2 Preparing the message for signing

The entire message M is taken as input M_1 to the computation of witness, $M_1 = M$ is the message; M_2 is empty, see 8.2 of ISO/IEC 14888-1.

B.2.3.3 Computing the witness

The deterministic witness is the hash token of the message, denoted H where H should be less than 2^{n-1}.

B.2.3.4 Computing the signature

The inputs to this stage are P and Q from the signature key X, K the randomizer computed in B.2.3.1.1, the pre-signature $\Pi = (U, V)$ computed in B.2.3.1.2 and the witness H computed in B.2.3.3. The signature S is computed using the formula:

$$S = K + (\lfloor (2^{2n}H - U)/PQ \rfloor V \bmod P)PQ \bmod N.$$

The output of this step is the signature $\Sigma = S$.

B.2.3.5 Constructing the appendix

The appendix is constructed from the signature and an optional text field, *text*. The text field could include a certificate which cryptographically ties the public verification key to the identification data of the signing entity.

B.2.3.6 Constructing the signed message

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

B.2.4 Verification process

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

B.2.4.1 Preparing message for verification

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

B.2.4.2 Retrieving the witness

The witness *H* is reconstructed from the data input M_1.

B.2.4.3 Computing the verification function

Using the integer *v* obtained either from the domain parameters ***Z*** or the verification key Y, the verifier computes $\overline{H}$, the high n bits of

$$S^v \bmod N.$$

B.2.4.4 Verifying the witness

The signature is valid only if the value of the reconstructed witness *H* agrees with value of the recomputed witness $\overline{H}$.

Annex C
(informative)

FIPS PUB 186 Generation of Primes P and Q

The prime generation scheme starts by using the SHA-1 and a user supplied SEED to construct a prime *Q*, in the range $2^{159} < Q < 2^{160}$. Once this is accomplished, the same SEED value is used to construct an X in the range $2^{L-1} < X < 2^{L}$. The prime *P* is then formed by rounding X to a number congruent to 1 mod 2*Q* as described below.

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

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

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

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

Note that the first bit of the sequence corresponds to the most significant bit of the corresponding integer and the last bit to the least significant bit.

Let *L*-1 = n*160 + b, where b and n are integers and $0 \leq b < 160$.

Step 1. Choose an arbitrary sequence of at least 160 bits and call it SEED. Let g be the length of SEED in bits.

Step 2. Compute *U* = SHA[SEED] XOR SHA[(SEED+1) mod 2^{g}].

Step 3. Form *Q* from *U* by setting the most significant bit (the 2^{159} bit) and the least significant bit to 1. In terms of Boolean operations, Q = *U* OR 2^{159} OR 1. Note that $2^{159} < Q < 2^{160}$.

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^{-80}.)

Step 5. If *Q* is not a prime, go to step 1.

Step 6. Let counter = 0 and offset = 2.

Step 7. For k = 0, … , n let V_k = SHA[(SEED + offset + k) mod 2^{g}].

Step 8. Let W be the integer $W = V_0 + V_1 * 2^{160} + \ldots + V_{n-1} * 2^{(n-1)*160} + (V_n \bmod 2^{b}) * 2^{n*160}$ and let $X = W + 2^{L-1}$. Note that $0 \leq W < 2^{L-1}$ and hence $2^{L-1} \leq X < 2^{L}$.

Step 9. Let c = X mod 2*Q* and set *P* = X - (c - 1). Note that *P* is congruent to 1 mod 2*Q*.

Step 10. If $P < 2^{L-1}$, then go to Step 13.

Step 11. Perform a robust primality test on *P*.

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

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

Step 14. If counter $\geq 2^{12}$ = 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*.

Annex D
(informative)

Elliptic Curve mathematical background

The text in this informative annex is extracted directly from ANSI X9.62-1998, Public Key Cryptography For the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). This text is included to provide additional elliptic curve mathematical background information to implementers beyond what is provided in the normative clauses of this standard. For even more elliptic curve mathematical information, see Menezes, A., «Elliptic Curve Public Key Cryptosystems.»

It is noted that some of the notation in this annex is slightly different than used elsewhere in this International Standard. For example, this annex will describe arithmetic with additive notation, whereas Clause A.2 uses multiplicative notation. As an example, gx converts to xG and ab converts to A+B.

D.1 Elliptic Curves and points

An elliptic curve E defined over $\boldsymbol{F}_q$ is a set of points $P = (x_p,y_p)$ where x_p and y_p are elements of $\boldsymbol{F}_q$ that satisfy a certain equation, together with the point at infinity denoted O. $\boldsymbol{F}_q$ is sometimes called the underlying field.

If $q = p$ is an odd prime (so the underlying field is $\boldsymbol{F}_p$) and $p>3$, then a,b shall satisfy $4a^3 + 27b^2 \neq 0$ (mod p), and every point $P = (x_p,y_p)$ on E (other than the point O) shall satisfy the following equation in $\boldsymbol{F}_p$:

$$y_p^2 + x_p^3 + ax_p + b.$$

If $q = 2^m$ is a power of 2 (so the underlying field is $\boldsymbol{F}_{2^m}$), then b shall be non-zero in $\boldsymbol{F}_{2^m}$, and every point $P = (x_P,y_P)$ on E (other than the point O) shall satisfy the following equation in $\boldsymbol{F}_{2^m}$:

$$y_p^2 + x_p y_p = x_p^3 + ax_p^2 + b.$$

An elliptic curve point P (which is not the point at infinity O) is represented by two field elements, the x-coordinate of P and the y-coordinate of P: $P = (x_P,y_P)$.

D.1.1 Addition rules for elliptic curves over $\boldsymbol{F}_p$

The set of points $E(\boldsymbol{F}_p)$ forms a group with the following addition rules:

(i) $O + O = O$

(ii) $(x,y) + O = O + (x,y) = (x,y)$ for all $(x,y) \in E(\boldsymbol{F}_p)$

(iii) $(x,y) + (x,-y) = O$ for all $(x,y) \in E(\boldsymbol{F}_p)$ (i.e. the negative of a the point (x,y) is -(x,y) = (x,-y))

(iv) (Rule for adding two distinct points that are not inverses of each other)

Let: $(x_1,y_1) \in E(\boldsymbol{F}_p)$ and $(x_2,y_2) \in E(\boldsymbol{F}_p)$ be two points such that $x_1 \neq x_2$.

Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

$x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$ and

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

(v) (Rule for doubling a point)

Let $(x_1,y_1) \in E(\boldsymbol{F}_p)$ be a point with $y_1 \neq 0$.

Then $2(x_1,y_1) = (x_3,y_3)$, where:

$x_3 = \lambda^2 - 2x_1$, $y_3 = \lambda(x_1 - x_3) - y_1$ and

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

The group $E(\boldsymbol{F}_p)$ is abelian, which means that $P_1 + P_2 = P_2 + P_1$ for all points P_1 and P_2 in $E(\boldsymbol{F}_p)$. The curve is said to be supersingular if $\# E(\boldsymbol{F}_p) = p + 1$; otherwise it is non-supersingular.

D.1.2 Addition rules for elliptic curves over $\boldsymbol{F}_{2^m}$

The set of points $E(\boldsymbol{F}_{2^m})$ forms a group with the following addition rules:

(i) $O + O = O$

(ii) $(x,y) + O = O + (x,y) = (x,y)$ for all $(x,y) \in E(\boldsymbol{F}_{2^m})$

(iii) $(x,y) + (x,x+y) = O$ for all $(x,y) \in E(\boldsymbol{F}_{2^m})$ (i.e. the negative of a the point (x,y) is $-(x,y) = (x,x+y)$)

(iv) (Rule for adding two distinct points that are not inverses of each other)

Let: $(x_1,y_1) \in E(\boldsymbol{F}_{2^m})$ and $(x_2,y_2) \in E(\boldsymbol{F}_{2^m})$ be two points such that $x_1 \neq x_2$.

Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$, $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ and $\lambda = \frac{y_1 + y_2}{x_1 + x_2}$

(v) (Rule for doubling a point)

Let $(x_1,y_1) \in E(\boldsymbol{F}_{2^m})$ be a point with $x_1 \neq 0$.

Then $2(x_1,y_1) = (x_3,y_3)$, where:

$x_3 = \lambda^2 + \lambda + a$, $y_3 = x_1^2 + (\lambda + 1)x_3$, and

$$\lambda = x_1 + \frac{y_1}{x_1}.$$

The group $E(\boldsymbol{F}_{2^m})$ is abelian, which means that $P_1 + P_2 = P_2 + P_1$ for all points P_1 and P_2 in $E(\boldsymbol{F}_{2^m})$.

Annex E
(informative)

Numerical examples of certificate-based digital signatures with appendix

E.1 The U.S. Digital Signature Algorithm (DSA)

A complete explanation of the generation of all values is given in FIPS PUB 186, Appendix 5. In this example the following values, expressed in hexadecimal notation, will be used.

E.1.1 DSA parameters

L = 200 (512_{10})

SEED = d5014e4b 60ef2ba8 b6211b40 62ba3224 e0427dd3

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

E.1.2 DSA 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

E.1.3 DSA per message data

K = 358dad57 1462710f 50e254cf 1a376b2b deaadfbf

K^{-1} = 0d516729 8202e49b 4116ac10 4fc3f415 ae52f917

M = ASCII form of "abc" = 616263

$h(M)$ = a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d

E.1.4 DSA signature

R = 8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0

S = 41e2345f 1f56df24 58f426d1 55b4ba2d b6dcd8c8

E.1.5 DSA Verification values

$\overline{R}$ = 8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0

E.2 The Pointcheval/Vaudenay Signature Algorithm

The following values are expressed in hexadecimal notation.

E.2.1 Pointcheval/Vaudenay parameters

L = 200 (=512_{10})

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

E.2.2 Pointcheval/Vaudenay 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

E.2.3 Pointcheval/Vaudenay per message data

K = 358dad57 1462710f 50e254cf 1a376b2b deaadfbf

K^{-1} = 0d516729 8202e49b 4116ac10 4fc3f415 ae52f917

M = ASCII form of "abc" = 616263

E.2.4 Pointcheval/Vaudenay signature

R = 8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0

$R \| M$ = 8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0 616263

$h(R \| M)$ = 2048680b 36d19516 cf78e869 beae7bc9 ab5dc543

S = 5bfdac3d 665fa38f 6ed315b3 b2f41b86 15187ccd

E.2.5 Pointcheval/Vaudenay Verification values

$\overline{\Pi}$ = 2fc6cb9a c3be0eac 3daf02ee fb96fca3 846708a2 8dd05730
165fe509 42f7f07e dfef8e52 fcb9369e 3814aa24 607e8047
5d0e61ad 461d6b16 b6cec5ba ae58946e

$\overline{R}$ = 8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0

E.3 Elliptic curve DSA

For the following examples, SHA-1 is used exclusively for the hash function, so that the hash token is simply the value of SHA-1, converted according to Annex C 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).

E.3.1 Example 1: Field F_2^m, m=191

E.3.1.1 Elliptic curve DSA parameters

The field F_2^{191} is represented as polynomials modulo the irreducible polynomial $x^{191} + x^9 + 1$.

The curve is E: $Y^2 + X Y = X^3 + aX^2 + b$ over F_2^{191}, where (in hexadecimal)

```
a  = 2866537b 67675263 6a68f565 54e12640 276b649e f7526267
b  = 2e45ef57 1f00786f 67b0081b 9495a3d9 5462f5de 0aa185ec
```

The base point is $G = (G_x, G_y)$ where (in hexadecimal)

```
Gx = 36b3daf8 a23206f9 c4f299d7 b21a9c36 9137f2c8 4ae1aa0d
Gy = 765be734 33b3f95e 332932e7 0ea245ca 2418ea0e f98018fb
```

The order of G is (in decimal)

Q = 1569275433846670190958947355803350458831205595451630533029.

E.3.1.2 Elliptic curve DSA signature key and verification key

The signature key is

X = 1275552191113212300012030439187146164646146646466749494799

The verification key is

$Y = G^X = (Y_x, Y_y)$ where (in hexadecimal)

```
Yx = 5de37e75 6bd55d72 e3768cb3o 96ffeb96 2614dea4 ce28a2e7
Yy = 55c0e0e0 2f5fb132 caf416ef 85b229bb b8e13520 03125ba1
```

E.3.1.3 Elliptic curve DSA per message data

M = ASCII form of "abc" = 616263

$h(M)$ = a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d

which is converted to an integer according to Annex C to get

H = 968236873715988614170569073515315707566766479517

The value of the randomizer is interpreted as an integer mod Q as

K = 1542725565216523985789236956265265265235675811949404040041

E.3.1.4 Elliptic curve DSA signature

$\Pi = G^K = (\Pi_x, \Pi_y)$ where (in hexadecimal)

```
Πx = 438e5a11 fb55e4c6 5471dcd4 9e266142 a3bdf2bf 9d5772d5
Πy = 2ad603a0 5bd1d177 649f9167 e6f475b7 e2ff590c 85af15da
```

$R = \Pi_x$, converted to an integer mod Q according to Annex C.

R = 87194383164871543355722284926904419997237591535066528048 mod Q

S = 308992691965804947361541664549085895292153777025772063598 mod Q

E.3.1.5 Elliptic curve DSA verification

The recomputed pre-signature is derived from the received message and verification key according to A.2.1.4.4.

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

$\overline{\Pi}_x$ = 438e5a11 fb55e4c6 5471dcd4 9e266142 a3bdf2bf 9d5772d5
$\overline{\Pi}_y$ = 2ad603a0 5bd1d177 649f9167 e6f475b7 e2ff590c 85af15da

The recomputed witness, $\overline{R}$ is $\overline{\Pi}_x$ converted to an integer mod Q

$\overline{R}$ = 871943831648715433557222849269044199972375915350665280 48

E.3.2 Example 2: Field F_p, 192-bit Prime p

E.3.2.1 Elliptic curve DSA parameters

The field is F_p where

p = 6277101735386680763835789423207666416083908700390324961279

The curve is E : $Y^2 = X^3+aX+b$ over F_p, where (in hexadecimal)

a = ffffffff ffffffff ffffffff fffffffe ffffffff fffffffc
b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

The base point is $G = (G_x, G_y)$ where (in hexadecimal)

G_x = 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
G_y = 07192b95 ffc8da78 631011ec 6b24cdd5 73f977a1 1e794811

The order of G is (in decimal)

Q = 6277101735386680763835789423176059013767194773182842284081

E.3.2.2 Elliptic curve DSA signature key and verification key

The signature key is selected randomly and kept secret. It's value as an integer mod Q is

X = 651056770906015076056810763456358567190100156695615665659

The corresponding verification key is given by

$Y = G^X = (Y_x, Y_y)$, where (in hexadecimal)

Y_x = 62b12d60 690cdcf3 30babab6 e69763b4 71f994dd 702d16a5
Y_y = 63bf5ec0 8069705f fff65e5c a5c0d697 16dfcb34 74373902

E.3.2.3 Elliptic curve DSA per message data

M = ASCII form of "abc" = 616263

$h(M)$ = a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d

which is converted to an integer according to Annex C to get

H = 968236873715988614170569073515315707566766479517

The value of the randomizer interpreted as an integer mod Q is given by

K = 6140507067065001063065065565667405560006161556565665656654

E.3.2.4 Elliptic curve DSA signature

$\Pi = G^K = (\Pi_x, \Pi_y)$ where (in hexadecimal)

Π_x = 88505238 0ff147b7 34c330c4 3d39b2c4 a89f29b0 f749fead
Π_y = 9cf9fa1c befefb91 7747a3bb 29c072b9 289c2547 884fd835

$R = \Pi_x$, converted to an integer mod Q according to Annex C.

$$R = 3342403536405981729393488334694600415596881826869351677613 \bmod Q$$

The signature, computed according to the signature function given in A.2.1 has value

$$S = 5735822328888155254683894997897571951568553642892029982342 \bmod Q$$

E.3.2.5 Elliptic curve DSA verification

The hash code is computed from the received message

$\overline{M}$ = ASCII form of "abc" = `616263`

$h(\overline{M})$ = `a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d`

and the recomputed hash token is $h(\overline{M})$ converted to an integer mod Q according to Annex C.

$$\overline{H} = 968236873715988614170569073515315707566766479517$$

The recomputed pre-signature is derived from the received message and verification key according to A.2.1.4.4. $\Pi = (\overline{\Pi}_x, \overline{\Pi}_y)$ where (in hexadecimal)

$\overline{\Pi}_x$ = `88505238 0ff147b7 34c330c4 3d39b2c4 a89f29b0 f749fead`
$\overline{\Pi}_y$ = `9cf9fa1c befefb91 7747a3bb 29c072b9 289c2547 884fd835`

The recomputed witness, $\overline{R}$ is $\overline{\Pi}_x$ converted to an integer mod Q

$$\overline{R} = 3342403536405981729393488334694600415596881826869351677613$$

The signature is verified since the recomputed witness equals the retrieved witness.

E.4 Digital signature with hashing based on ISO/IEC 9796

In this example, all values are presented in hexadecimal. This example is derived from U.S. ANSI Standard X9.31. The examples in X9.31 use a nonstandard representation of integers mod N. To be consistent with this standard, those integer values are modified.

E.4.1 Example with v odd (v = 3)

E.4.1.1 Generation of signature key and verification key

E.4.1.1.1 Public verification exponent

$v = 3$

E.4.1.1.2 Private signature key

E.4.1.1.2.1 Private prime factors

P_1 =	d8cd81f0	35ec57ef	e8229551	49d3bff7	0c53520d
	769d6d76	646c7a79	2e16ebd8	9fe6fc5b	6060bd97
	8ed64a90	59c5b039	98a0e94c	86d78b85	ba37b5af
	d987505f				

P_2 =	cc109249	5d867e64	065dee3e	7955f2eb	c7d47a2d
	7c995338	8f97dddc	3e1ca19c	35ca659e	dc3d6c08
	f64068ea	fedbd911	27f9cb7e	dc174871	1b624e30
	b857caad				

E.4.1.1.2.2 Private signature exponent

s =	1ccda20b	cffb8d51	7ee96668	66621b11	822c7950
	d55f4bb5	bee37989	a7d17312	e326718b	e0d62ccb
	11415f78	b36be2e6	0d599d4e	41346c82	d845498a
	81b2f663	2fd7d1cc	efcabf74	17350238	109ec289
	d5382762	b77a1c99	96dd1d2b	71a52faf	52aba9de
	d19f3f5d	5d71d054	73ec9c79	92d84128	0bac72b8
	7bf51eb1	ccb65c87			

E.4.1.1.3 Public verification modulus

N =	acd1cc46	dfe54fe8	f9786672	664ca269	0d0ad7e5
	003bc642	7954d939	eee8b271	52e6a947	45050cc2
	67883cd4	34875164	5019afd5	873a8b11	119fb93f
	0a31c654	c3ecff07	3233530c	79be90e0	26e2421d
	d378b88b	40136c48	7d33075a	1612ab90	c5b75d33
	2659a5d0	b5c19576	102d3424	31ac3bbb	a8f98449
	bd58bc0b	5e254633			

E.4.1.2 Signature Generation

M = ASCII form of "abc" = 616263

h(*M*) =	a9993e36	4706816a	ba3e2571	7850c26c	9cd0d89d

The string `x'33cc'` is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble `x'b'` (and where the rightmost trailing nibble is always the hexadecimal value `x'a'` acting as a field separator to the hash *h*(*M*)) is prefixed to the hash. This is preceded by the header hex value `x'6'`.

H =	6bbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbaa999	3e364706	816aba3e	25717850
	c26c9cd0	d89d33cc			

$S = H^s \bmod N$ is now computed to be the following:

S =	500abdc2	48d78e2d	b9182a98	7b296e93	53083435
	070fbe16	b1629a30	7cab53d3	c9b70611	bffa479e
	cb744397	b01c6f1c	b4775051	1510005e	e9f83709
	15788172	98db07fb	b746c6d7	774bb069	64244463
	3abc79c2	0cb81f8b	df9ff07e	eba2efc3	11a80438
	622492c8	89fc0b17	4681e5ce	427149c9	8fe34580
	5112f4d2	d8b53761			

E.4.1.3 Signature Verification

The value $\overline{H} = S^v \bmod (N)$ is computed.

$\overline{H}$ =	6bbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbaa999	3e364706	816aba3e	25717850
	c26c9cd0	d89d33cc			

Since $\overline{H} = H \bmod (N)$, the signature is verified.

E.4.2 Example with v even (v = 2)

The padding used in U.S. ANSI Standard X9.31 ensures that any valid witness ends with hexadecimal value of `x'c'` or `x'6'`.

E.4.2.1 Generation of signature key and verification key

E.4.2.1.1 Public verification exponent

$v = 2$

E.4.2.1.2 Private signature key

E.4.2.1.2.1 Private prime factors

P_1 =	dbb3cb4c	375c0ecd	2fd300db	4f085472	93ca004c
	edd2019c	e79ca08a	15eefb25	dd3baf98	183b0c2f
	01d7f8b4	931856f6	dd3eba17	7d763c03	e1dceabc
	d803be33				

P_2 =	eeaa4a53	47999fe7	6fb78760	64bbec66	cb409a77
	39ef5a59	06613dc3	7225d41d	2beb1f9f	5ec77a85
	38767a87	bb7015d6	07ff26de	61282753	9306ba1c
	fff093a7				

E.4.2.1.2.2 Private signature exponent

s =	199a6985	e9b2bff5	a2841ccc	d80fc73a	28a14266
	0987eb12	3dbcaeb2	b8ee546d	2356a3a5	7d9c28ed
	71e455c4	466cbe30	7787dc5a	9959b747	5a189a8f
	038a4741	e4b10153	be08c26e	4401f7ab	6e7e9609
	2caf07c0	870b13b6	4f669667	3029ec2c	77aabc39
	7fa528a2	45d7073c	e69cc9bd	cd7bef91	599dca48
	4000c0bd	8ab0814e			

E.4.2.1.3 Public verification modulus

N =	ccd34c2f	4d95ffad	1420e666	c07e39d1	450a1330
	4c3f5891	ede57595	c772a369	1ab51d2b	ece1476b
	8f22ae22	3365f183	bc3ee2d4	cacdba3a	d0c4d478
	1c523a10	efe6203d	6f3bc226	bf9a4597	27b8f122
	c482d8c8	6019f9a8	69329187	096430a6	c67cb103
	742bcbc6	6906ad23	836ebabb	511d5d80	ab8cb599
	74e9aac6	2d785c45			

E.4.2.2 Signature Generation

M = ASCII form of "abc" = 616263

h(M) =	a9993e36	4706816a	ba3e2571	7850c26c	9cd0d89d

The string `x'33cc'` is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble `x'b'` (and where the rightmost trailing nibble is always the hexadecimal value `x'a'` acting as a field separator to the hash $h(M)$) is prefixed to the hash. This is preceded by the header hex value `x'6'` to give an intermediate value H'.

H' =	6bbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb	bbbbbbbb
	bbbbbbbb	bbbaa999	3e364706	816aba3e	25717850
	c26c9cd0	d89d33cc			

Since the Jacobi symbol $\left(\frac{H'}{N}\right)$ of H' with respect to n is -1, H' is divided by 2 before signing in order to force the Jacobi symbol of $H = H'/2$ to +1.

$H =$	35dddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddd54cc	9f1b2383	40b55d1f	12b8bc28
	61364e68	6c4e99e6			

$S = H^s \bmod N$ is now computed to be the following:

$S =$	232f0e08	eb9a2395	7646697f	c7884796	d39a04fd
	0eff5b72	b60813d4	e6919178	91c96603	876d0879
	3aad86da	f2e6187f	f62c226e	81bd6b99	3b27091e
	0864895a	f10f222a	eb022961	b444d312	ea3db789
	1d4550b2	80cf2469	3d4465b9	57e53cbd	b0f8c29d
	2b5ee154	5d6c91a4	5eaaacec	0096d8a5	e4cfe06a
	2cd320bd	f853d817			

E.4.2.3 Signature Verification

The intermediate value $\overline{H'} = S^v \bmod (N)$ is computed.

$\overline{H'} =$	96f56e51	6fb821cf	36430888	e2a05bf3	672c3552
	6e617ab4	100797b7	e994c58b	3cd73f4e	0f03698d
	b144d044	558813a5	de6104f6	ecefdc5c	f2e6f69a
	3e745c33	1208425f	915de448	e1bc67b9	49db1344
	e6a4faea	823c1bca	8b54b3a9	2b8652c8	e89ed325
	964dede8	8b295856	e4539738	10680061	98d3f971
	13b35c5d	c129c25f			

Since $\overline{H'}$ ends with the hexadecimal value x'f' which is not a valid witness value of x'c' or x'6', $\overline{H} = (N - \overline{H'})$

$\overline{H} =$	35dddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddddddd	dddddddd	dddddddd	dddddddd
	dddddddd	dddd54cc	9f1b2383	40b55d1f	12b8bc28
	61364e68	6c4e99e6			

the signature is verified

E.5 ESIGN signature algorithm

E.5.1 ESIGN domain parameters

n = 768

s = v = 1024

E.5.2 Signature key and verification key

E.5.2.1 Signature key

The signature key consists of two primes P and Q whose values (given in hexadecimal) are

$P_1=P_2=P$ =	fd3764f3	7b98dfe4	8e30b2c4	004e2d03
	0a5e8018	2f94b156	fe6e4b5f	16f902da
	d60e4730	30deab98	75f3d749	de79c361
	8874d195	4102dfe0	47637bab	495c7dc2
	912fdeb9	4b2d5eca	b798e90e	c6e634b7
	b4f1153b	4d9f4bd0	3c45cfc7	2600e549
$P_3=Q$ =	8332d671	713a0dea	71e9453a	b323c499
	2455d957	ef6985a5	3770af04	e1c76529
	a0bc855e	ca025f9c	540cf0b5	3684ea5e
	5777b647	17e78b99	1c2bacb6	9befed40
	f414d805	a1594e56	90ce67f6	42c42714
	7c94ba1f	2dc9adf8	eacd114b	1723700f

E.5.2.2 Verification key

The verification key consists of the verification exponent,

$v = s = 1024$

and the modulus, whose value (given in hexadecimal) is

N =	805c6554	66eea57c	a1798241	5aa1aca7
	df54ab5c	17953109	9a08cf05	5d6bd99f
	7e5d4ff8	95cb633b	3368dac6	8c3ff751
	1c5ccf45	6ade1aa2	20558dad	17d466df
	f0e7f3b9	3ddd6934	07a18a66	bc74ceb1
	ebac6901	4b6ce22a	78e70676	4ca5de4d
	196c7007	54cb46c7	30f77bc0	bc1955cc
	fb26df7e	4c005dc7	b836acc2	f04e696b
	10578b6d	2cb993f3	4a01fb95	2727517f
	4ac8499a	51829133	16b2fcaa	5c594c3e
	9b8b24ec	313c8863	4b7bbfef	bfdac7eb
	689c79a8	6b5c4401	b7ece53c	ab9f2326
	25c70842	2f5fe450	9631128d	a2775427
	0af91fc9	b09800a0	e4339609	aa9a10b6
	2f6812f1	91a3d598	177001a0	88db58a4
	ad2fef5a	230735e0	0aeb8031	50403d11
	51f15167	65bada30	d57f2b4c	b9438e59
	551828f1	9704aab5	4169f107	e66dae3f

E.5.3 ESIGN signature process

E.5.3.1 ESIGN pre-signature

E.5.3.1.1 ESIGN randomizer

The randomizer is given (in hexadecimal) by

K =	76a4d0dd	5b024775	2d546ca4	27b6e8be