INTERNATIONAL STANDARD

ISO/IEC 14888-3

Second edition
2006-11-15

AMENDMENT 1
2010-06-15

Information technology — Security techniques — Digital signatures with appendix —

Part 3: Discrete logarithm based mechanisms

AMENDMENT 1: Elliptic Curve Russian Digital Signature Algorithm, Schnorr Digital Signature Algorithm, Elliptic Curve Schnorr Digital Signature Algorithm, and Elliptic Curve Full Schnorr Digital Signature Algorithm

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

Partie 3: Mécanismes basés sur un logarithme discret

AMENDEMENT 1: Algorithme de signature numérique russe de courbe elliptique, algorithme de signature numérique schnorr, algorithme de signature numérique schnorr de courbe elliptique, et algorithme de signature numérique schnorr totale de courbe elliptique

Reference number
ISO/IEC 14888-3:2006/Amd.1:2010(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

Foreword

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

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

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

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

Amendment 1 to ISO/IEC 14888-3:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

Amendment 1 to ISO/IEC 14888-3:2006 introduces four digital signature algorithms: Elliptic Curve Russian Digital Signature Algorithm (EC-RDSA), Schnorr Digital Signature Algorithm (SDSA), Elliptic Curve Schnorr Digital Signature Algorithm (EC-SDSA), and Elliptic Curve Full Schnorr Digital Signature Algorithm (EC-FSDSA). Object identifiers, test vectors, a comparison of certificate-based mechanisms, and claimed features for choosing a mechanism are given.

Information technology — Security techniques — Digital signatures with appendix —

Part 3: Discrete logarithm based mechanisms

AMENDMENT 1: Elliptic Curve Russian Digital Signature Algorithm, Schnorr Digital Signature Algorithm, Elliptic Curve Schnorr Digital Signature Algorithm, and Elliptic Curve Full Schnorr Digital Signature Algorithm

Page 1, Clause 2

In the second normative reference, replace “1998” with “2008”.

Page 3, Clause 4

Add the following to the end of the list of symbols:

Π_Y	y-value of Π
$\boldsymbol{F}$	a finite field
F_P	a finite field of prime order p
Z_P^*	a multiplicative group over F_P

Page 27, Clause 6

Add the following subclauses after 6.6.4.6:

6.7 EC-RDSA

6.7.1 Introduction to EC-RDSA

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

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

where $(T_1, T_2) = (H, R)$ and $H = h(M)$ is the hash-code of message M, converted to an integer as described in 6.7.4.5.

The witness function is defined by the formula

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

Thus the signature equation becomes

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

NOTE EC-RDSA stands for Elliptic Curve Russian Digital Signature Algorithm. The mechanism is taken from a Russian State Standard [36]. The notation here has been changed from GOST R 34.10-2001 to conform with the notation used in ISO/IEC 14888.

6.7.2 Parameters

p	a prime
$\boldsymbol{E}$	an elliptic curve group over the field $GF(p)$
$\#\boldsymbol{E}$	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.

6.7.3 Generation of signature key and verification key

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

$$Y = [X]G.$$

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

NOTE The Russian standard for digital signature (GOST R 34.10-2001) does not completely specify the process of generation of a user's secret signature key X.

6.7.4 Signature process

6.7.4.1 Producing the randomizer

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

6.7.4.2 Producing the pre-signature

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

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

6.7.4.3 Preparing message for signing

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

6.7.4.4 Computing the witness

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

6.7.4.5 Computing the assignment

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

6.7.4.6 Computing the second part of the signature

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

$S = RX + KH \bmod q$.

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

6.7.4.7 Constructing the appendix

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

6.7.4.8 Constructing the signed message

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

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

6.7.5 Verification process

6.7.5.1 Retrieving the witness

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

6.7.5.2 Preparing message for verification

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

6.7.5.3 Retrieving the assignment

This stage is identical to 6.7.4.5. The inputs to the assignment function consist of the witness R from 6.7.5.1 and M_2 from 6.7.5.2. The assignment $T = (T_1, T_2)$ is recomputed as the output from the assignment function given in 6.7.4.5.

6.7.5.4 Recomputing the pre-signature

The inputs to this stage are system parameters, the verification key Y, the assignment $T = (T_1, T_2)$ from 6.7.5.3, and the second part of the signature S from 6.7.5.1. The verifier obtains a recomputed value $\bar{\Pi}$ of the pre-signature by computing it using the formula

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

6.7.5.5 Recomputing the witness

The computations at this stage are the same as in 6.7.4.4. The verifier executes the witness function. The input is $\bar{\Pi}$ from 6.7.5.4. The output is the recomputed witness $\bar{R}$.

6.7.5.6 Verifying the witness

The verifier compares the recomputed witness, $\bar{R}$ from 6.7.5.5 to the retrieved version of R from 6.7.5.1. If $\bar{R} = R$, then the signature is verified.

6.8 SDSA

6.8.1 Introduction to SDSA

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

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

Thus the signature equation becomes

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

6.8.2 Parameters

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

Four choices for the pair (α, h) are allowed in SDSA, namely (1024, SHA-1), (2048, SHA-224), (2048, SHA-256), and (3072, SHA-256). Corresponding β should be selected according to α in 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. The parameters p and q can be generated using the prime generation techniques given in ISO/IEC 18032.

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

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.

6.8.3 Generation of signature key and verification key

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

6.8.4 Signature process

6.8.4.1 Producing the randomizer

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

6.8.4.2 Producing the pre-signature

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

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

6.8.4.3 Preparing message for signing

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

6.8.4.4 Computing the witness

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

$$R = h(\Pi || M).$$

6.8.4.5 Computing the assignment

The witness R is converted to an integer according to conversion rule, *BS2I,* in Annex B and then reducing modulo q. The assignment (T_1, T_2) is $(-1, -BS2I(R) \bmod q)$.

6.8.4.6 Computing the second part of the signature

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

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

6.8.4.7 Constructing the appendix

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

6.8.4.8 Constructing the signed message

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

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

6.8.5 Verification process

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

6.8.5.1 Retrieving the witness

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

6.8.5.2 Preparing message for verification

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

6.8.5.3 Retrieving the assignment

The input to the assignment function consists of the witness *R* from 6.8.5.1. The assignment

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

6.8.5.4 Recomputing the pre-signature

The inputs to this stage are domain parameters, verification key *Y*, assignment $T = (T_1, T_2)$ from 6.8.5.3 and second part of the signature *S* from 6.8.5.1. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

$\Pi' = Y^{(T_2 \bmod q)} G^{(-ST_1 \bmod q)} \bmod p$.

6.8.5.5 Recomputing the witness

The computations at this stage are the same as in 6.8.4.4 and 6.8.4.5. The verifier executes the witness function. The input is Π' from 6.8.5.4 and M_1 from 6.8.5.2. The output is the recomputed witness *R*' which is the hash-code of the recomputed pre-signature Π' and the message *M*.

6.8.5.6 Verifying the witness

The verifier compares the recomputed witness, R' from 6.8.5.5 to the retrieved version of *R* from 6.8.5.1. If $R' = R$, then the signature is verified.

6.9 EC-SDSA

6.9.1 Introduction to EC-SDSA

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

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

NOTE This specification of EC-SDSA generates the witness by hashing the concatenation of the x-coordinate, the y-coordinate and the message. However, the mechanism would remain secure even if the y-coordinate was omitted from the hash computation. As a result, future versions of this standard may permit the omission of the y-coordinate from the hash calculation to improve performance.

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

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

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

Thus the signature equation becomes:

$$-K + T_2 X + S \equiv 0 \pmod{q}.$$

6.9.2 Parameters

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

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

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

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

6.9.3 Generation of signature key and verification key

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

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

6.9.4 Signature process

6.9.4.1 Producing the randomizer

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

6.9.4.2 Producing the pre-signature

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

6.9.4.3 Preparing message for signing

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

6.9.4.4 Computing the witness

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

6.9.4.5 Computing the assignment

The witness R is converted to an integer according to conversion rule, *BS2I,* in Annex B and then reducing modulo q.
The assignment (T_1, T_2) is $(-1, -BS2I(R) \bmod q)$.

6.9.4.6 Computing the second part of the signature

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

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

6.9.4.7 Constructing the appendix

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

6.9.4.8 Constructing the signed message

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

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

6.9.5 Verification process

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

6.9.5.1 Retrieving the witness

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

6.9.5.2 Preparing message for verification

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

6.9.5.3 Retrieving the assignment

The input to the assignment function consists of the witness R from 6.9.5.1. The assignment $T = (T_1, T_2) = (-1, -BS2I(R) \bmod q)$.

6.9.5.4 Recomputing the pre-signature

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

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

6.9.5.5 Recomputing the witness

The computations at this stage are the same as in 6.9.4.4 and 6.9.4.5. The verifier executes the witness function from 6.9.4.4. The input is Π' from 6.9.5.4. The output is the recomputed witness R' which is the hash-code of the recomputed pre-signature Π' and the message M.

6.9.5.6 Verifying the witness

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

6.10 EC-FSDSA

6.10.1 Introduction to EC-FSDSA

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

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

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

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

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

Thus, the signature equation becomes

$$-K + T_2 X + S \equiv 0 \pmod q.$$

6.10.2 Parameters

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

Hash function identifier or OID with specified hash function.

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

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

6.10.3 Generation of signature key and verification key

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

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

6.10.4 Signature process

6.10.4.1 Producing the randomizer

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

6.10.4.2 Producing the pre-signature

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

6.10.4.3 Preparing message for signing

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

6.10.4.4 Computing the witness

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

6.10.4.5 Computing the assignment

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

6.10.4.6 Computing the second part of the signature

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

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

6.10.4.7 Constructing the appendix

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

6.10.4.8 Constructing the signed message

A signed message is the concatenation of the message, M, and the appendix:
$(M \,\|\, (R,S) \,\|\, text)$.

6.10.5 Verification process

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

6.10.5.1 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 R represents a point on E and $0 < S < q$; if either condition is violated the signature shall be rejected.

6.10.5.2 Preparing message for verification

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

6.10.5.3 Retrieving the assignment

The input to the assignment function is computed as in 6.10.4.5 from the witness R from 6.10.4.4 and the message M from 6.10.4.3. The assignment is given by $T = (T_1,T_2) = (-1, -BS2I(h(R||M)) \bmod q)$.

6.10.5.4 Recomputing the pre-signature

The inputs to this stage are system parameters, verification key Y, assignment $T = (T_1,T_2)$ from 6.10.5.3 and second part of the signature S from 6.10.5.1. The verifier obtains a recomputed value Π' of the pre-signature by computing it using the formula

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

6.10.5.5 Recomputing the witness

The computations at this stage are the same as in 6.10.4.4. The verifier executes the witness function. The input is Π' from 6.10.5.4. The output is the recomputed witness R'.

6.10.5.6 Verifying the witness

The verifier compares the recomputed witness, R' from 6.10.5.5 to the retrieved version of R from 6.10.5.1. If $R' = R$, then the signature is verified.

Page 33, Annex A

Replace all text after the first paragraph with the following:

```
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) }
id-dswa-dl-EC-RDSA  OID ::= { id-dswa-dl ec-rdsa(9) }
id-dswa-dl-SDSA     OID ::= { id-dswa-dl sdsa(10) }
id-dswa-dl-EC-SDSA  OID ::= { id-dswa-dl ec-sdsa(11) }
id-dswa-dl-EC-FSDSA  OID ::= { id-dswa-dl ec-fsdsa(12) }



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

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


   ...  -- Expect additional algorithms --
}

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

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

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

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

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

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

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

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

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

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

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

```
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 3 - 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)}
```

Page 63, Annex F

Add the following subclauses to the end of Annex F:

F.9 EC-RDSA mechanism

F.9.1 Introduction to the EC-RDSA mechanism

For the following example, SHA-256 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-256, converted according to Annex B to the appropriate data item.

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.9.2 Field F_P, 256-bit Prime P

F.9.2.1 Parameters

The field is F_P where P is in hexadecimal

P = 80000000 00000000 00000000 00000000 00000000 00000000
00000000 00000431

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

a = 7

b = 5FBFF498 AA938CE7 39B8E022 FBAFEF40 563F6E6A 3472FC2A
514C0CE9 DAE23B7E

$G = (G_X, G_Y)$

G_X = 2

G_Y = 08E2A8A0 E65147D4 BD631603 0E16D19C 85C97F0A 9CA26712 2B96ABBC EA7E8FC8

Q = 80000000 00000000 00000000 00000001 50FE8A18 92976154 C59CFC19 3ACCF5B3

F.9.2.2 Signature key and verification key

X = 7A929ADE 789BB9BE 10ED359D D39A72C1 1B60961F 49397EEE 1D19CE98 91EC3B28

$Y = (Y_X, Y_Y)$

Y_X = 7F2B49E2 70DB6D90 D8595BEC 458B50C5 8585BA1D 4E9B788F 6689DBD8 E56FD80B

Y_Y = 26F1B489 D6701DD1 85C8413A 977B3CBB AF64D1C5 93D26627 DFFB101A 87FF77DA

F.9.2.3 Per message data

M = ASCII form of "abc" = 616263

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

K = 77105C9B 20BCD312 2823C8CF 6FCC7B95 6DE33814 E95B7FE6 4FED9245 94DCEAB3

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

Π_X = 41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8 3AD043FD 39DC0493

Π_Y = 489C375A 9941A304 9E33B343 61DD2041 72AD98C3 E5916DE2 7695D22A 61FAE46E

F.9.2.4 Signature

R = 41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8 3AD043FD 39DC0493

S = 0A7BA472 2DA5693F 229D175F AB6AFB85 7EC2273B 9F88DA58 92CED311 7FCF1E36

F.9.2.5 Verification

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

$\bar{\Pi}_x$ = 41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8 3AD043FD 39DC0493

$\bar{\Pi}_y$ = 489C375A 9941A304 9E33B343 61DD2041 72AD98C3 E5916DE2 7695D22A 61FAE46E

$\bar{R}$ = 41AA28D2 F1AB1482 80CD9ED5 6FEDA419 74053554 A42767B8 3AD043FD 39DC0493

F.10 SDSA mechanism

F.10.1 Introduction to the SDSA mechanism

For example F.10.2, the RFC-5114 2048-bit MODP group with 224-bit prime order subgroup is used as the group. SHA-224 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-224, converted according to Annex B to the appropriate data item.

For example F.10.3, the RFC-5114 2048-bit MODP group with 256-bit prime order subgroup is used as the group. SHA-256 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-256, converted according to Annex B to the appropriate data item.

F.10.2 2048-bit Group with 224-bit Prime order Subgroup

F.10.2.1 Parameters

α = 2048

β = 224

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

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

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

F.10.2.2 Signature key and verification key

X = 602FE736 80BEFCB2 A8B46779 35FF652B 21A3F4DE 46725D07
D7D371A9

Y = A7DBB446 FD8C4B82 61BE026D 94FA9847 74B17110 CABC0944
14AD2013 2EFC8B7D 7C7FF05D D0B902C4 EF736831 61C1F9A3
9D60E7AB E3BD9FE2 B458A96D F4783408 0AA93CAF 09673967
F434548D 44B278E0 4C1FA6D5 E0C41990 CEF37940 66015ED4
748DAD56 429596DD 9259C45C 21B71A5E A4EF099A 06DAC737
8958A107 B11B3E57 384118E0 19897C48 E734F069 E717E23A
DD202405 823A2AE7 A08AFA51 09D2CF6A 0FF546FA 38A8735D
1CE715E0 6AC08EB0 93EB331F EBEC88D6 1DF546E2 DC9E8465
10B63F6A 5BA73FE3 6995BD17 1B9D7D35 9EEB3D7D B801F382
1D582280 DF71A27D 5191E4AB 42BE27A3 1180F537 CABAC0D1
2EBF1698 B1884697 9EDCA15D DC8F86DC

F.10.2.3 Per message data

M = ASCII form of "abc" = 616263

K = 7BFA2DD5 6B31BB27 FFC0D1AE 1ABAA90F A0BB9379 08A542A1
5EFD1E15

Π = 60FCB613 40799851 B5E2DC3A 3865BC21 29100D38 4B1C9A94
6F0C873B 442BEBD8 5904CD09 A4C6A29E 0CD1111E B9E65F82
85F8A578 A5717098 FA2A601F D9183CDD D5FF1586 AB255E1D
4DF4A141 DFE717DC 16DA3B0D 438B1EA5 4976523F 1D73351B
F39B1987 97DA0EC7 E9EE994A 4C0352D8 271D186A 0DEA8AB0
FD5E7862 17016E91 03C5F139 2C1D3C01 B974BADC 88184905
065F8DA8 55656BAF B3B1EDBC 4C14A969 2AEA1A71 D85117F4
08548EF5 A34966B9 0123FC81 72472B44 06D0C2E6 77C3C21D
D0680C63 0DC69BFF BA67D89F F17CA52A 6B0F164F 5452777D
B838BDBD EE60E03B AE773475 42435BD9 09D021DD F97602E0
3FE41463 CC18128B AFA6E661 6F6CA744

R = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3
316AD681

F.10.2.4 Signature

R = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3
316AD681

S = 4C776699 9F9D52E1 52B47F29 335E548F A3B90625 C55FC9A4
FE2D5F1F

F.10.2.5 Verification

Π' = 60FCB613 40799851 B5E2DC3A 3865BC21 29100D38 4B1C9A94
6F0C873B 442BEBD8 5904CD09 A4C6A29E 0CD1111E B9E65F82
85F8A578 A5717098 FA2A601F D9183CDD D5FF1586 AB255E1D
4DF4A141 DFE717DC 16DA3B0D 438B1EA5 4976523F 1D73351B
F39B1987 97DA0EC7 E9EE994A 4C0352D8 271D186A 0DEA8AB0
FD5E7862 17016E91 03C5F139 2C1D3C01 B974BADC 88184905
065F8DA8 55656BAF B3B1EDBC 4C14A969 2AEA1A71 D85117F4
08548EF5 A34966B9 0123FC81 72472B44 06D0C2E6 77C3C21D
D0680C63 0DC69BFF BA67D89F F17CA52A 6B0F164F 5452777D
B838BDBD EE60E03B AE773475 42435BD9 09D021DD F97602E0
3FE41463 CC18128B AFA6E661 6F6CA744

R' = CC192C12 72872367 48346281 4DF721E5 D9B2A651 0D97F3D3
316AD681

F.10.3 2048-bit MODP Group with 256-bit Prime order Subgroup

F.10.3.1 Parameters

α = 2048

β = 256

p = 87A8E61D B4B6663C FFBBD19C 65195999 8CEEF608 660DD0F2
5D2CEED4 435E3B00 E00DF8F1 D61957D4 FAF7DF45 61B2AA30
16C3D911 34096FAA 3BF4296D 830E9A7C 209E0C64 97517ABD
5A8A9D30 6BCF67ED 91F9E672 5B4758C0 22E0B1EF 4275BF7B
6C5BFC11 D45F9088 B941F54E B1E59BB8 BC39A0BF 12307F5C
4FDB70C5 81B23F76 B63ACAE1 CAA6B790 2D525267 35488A0E
F13C6D9A 51BFA4AB 3AD83477 96524D8E F6A167B5 A41825D9
67E144E5 14056425 1CCACB83 E6B486F6 B3CA3F79 71506026
C0B857F6 89962856 DED4010A BD0BE621 C3A3960A 54E710C3
75F26375 D7014103 A4B54330 C198AF12 6116D227 6E11715F
693877FA D7EF09CA DB094AE9 1E1A1597

G = 3FB32C9B 73134D0B 2E775066 60EDBD48 4CA7B18F 21EF2054
07F4793A 1A0BA125 10DBC150 77BE463F FF4FED4A AC0BB555
BE3A6C1B 0C6B47B1 BC3773BF 7E8C6F62 901228F8 C28CBB18
A55AE313 41000A65 0196F931 C77A57F2 DDF463E5 E9EC144B
777DE62A AAB8A862 8AC376D2 82D6ED38 64E67982 428EBC83
1D14348F 6F2F9193 B5045AF2 767164E1 DFC967C1 FB3F2E55
A4BD1BFF E83B9C80 D052B985 D182EA0A DB2A3B73 13D3FE14
C8484B1E 052588B9 B7D2BBD2 DF016199 ECD06E15 57CD0915
B3353BBB 64E0EC37 7FD02837 0DF92B52 C7891428 CDC67EB6
184B523D 1DB246C3 2F630784 90F00EF8 D647D148 D4795451
5E2327CF EF98C582 664B4C0F 6CC41659

q = 8CF83642 A709A097 B4479976 40129DA2 99B1A47D 1EB3750B
A308B0FE 64F5FBD3

F.10.3.2 Signature key and verification key

X = 73018895 20D47AA0 55995BA1 D8FCD701 6EA62E09 18892E07
B7DC23AF 69006B88

Y = 57A17258 D4A3F47C 4545AD51 F3109C5D B41B7878 79FCFE53
8DC1DD5D 35CF42FF 3A9F225E DE650212 6408FCB1 3AEA2231
80B149C4 64E176EB F03BA651 0D8206C9 20F6B1E0 9392E6C8
40A05BDB 9D6875AB 3F4817EC 3A65A665 B788ECBB 447188C7
DF2EB4D3 D9424E57 D964398D BE1C6362 659C6BD8 55C1D3E5
1D64796C A598480D FDD9580E 55085345 C15E34D6 A33A2F43
E222407A CE058972 D34952AE 2B705C53 2243BE39 4B222329
6161145E F2927CDB C55BBD56 4AAE8DE4 BA4500A7 FA432FE7
8B0F0689 1E408083 7E761057 BC6CB8AC 18FD4320 7582032A
FB63C624 F32E66B0 5FC31C5D FFB25FA9 2D4D00E2 B0D4F721
E88C417D 2E57797B 8F55A2FF C6EE4DDB

F.10.3.3 Per message data

M = ASCII form of "abc" = 616263

K = 2B73E8FF 3A7C0168 6CA556E0 FABFD74A C8D1FDA4 AD3D503F
23B8EB8A EEC63305

Π = 41979DBA 19606871 A25BBB51 665AD584 9511D125 8C077E93
027D79AC 35EE887C 460C2689 3D7FFC59 0F317B7A 2C01FCB4
3667E373 F8F2D239 0C950BAF 972E6E0B 00A79416 9696CC95
08A895D6 3F8AA7EA 564AA5DE 6104920A 5E9F687F 469BC831
0C02BE30 B8FD4A54 A167E114 01FEE171 EF284811 6146CC69
C0899526 7186C43E 98B5E62E 9CE5C344 646950EF F57B1490
27FE078F CD9C8297 4AFF1DDE 5AF18E4C A3E3787F 66D15D23
292B5F4E 225AAC64 FC904AB3 40A88B76 40F2D436 D2840185
077EACA4 EE75113E 95B26149 F7C6D2CD 554463F9 26E48F09
AD3C99B8 5EA3EC39 E795FEAE C90C8293 FB0D0506 0FE2BF91
5F00E2FB 7C17B2E8 7C462ED0 D49B8E2F

R = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3
D6DB0CE6 E0ED696E

F.10.3.4 Signature

R = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3
D6DB0CE6 E0ED696E

S = 3505AEA2 E039E18F DDC6580A E89E15DF 0103FB45 C1BB763E
DA4EE6F5 F01783CE

F.10.3.5 Verification

Π' = 41979DBA 19606871 A25BBB51 665AD584 9511D125 8C077E93
027D79AC 35EE887C 460C2689 3D7FFC59 0F317B7A 2C01FCB4
3667E373 F8F2D239 0C950BAF 972E6E0B 00A79416 9696CC95
08A895D6 3F8AA7EA 564AA5DE 6104920A 5E9F687F 469BC831
0C02BE30 B8FD4A54 A167E114 01FEE171 EF284811 6146CC69
C0899526 7186C43E 98B5E62E 9CE5C344 646950EF F57B1490
27FE078F CD9C8297 4AFF1DDE 5AF18E4C A3E3787F 66D15D23
292B5F4E 225AAC64 FC904AB3 40A88B76 40F2D436 D2840185
077EACA4 EE75113E 95B26149 F7C6D2CD 554463F9 26E48F09
AD3C99B8 5EA3EC39 E795FEAE C90C8293 FB0D0506 0FE2BF91
5F00E2FB 7C17B2E8 7C462ED0 D49B8E2F

R' = CDAC932A 758FCFCE 7E549903 FD891F41 FB5410CB DDD246F3
D6DB0CE6 E0ED696E

F.11 EC-SDSA mechanism

F.11.1 Introduction to the EC-SDSA mechanism

For example F.11.2, the NIST curve P256 is used as the elliptic curve. SHA-256 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-256, converted according to Annex B to the appropriate data item.

For example F.11.3, the NIST curve P384 is used as the elliptic curve. SHA-384 is used exclusively for the hash-function, so the hash-code is simply the value of SHA-384, 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.11.2 Field F_P, 256-bit Prime P

F.11.2.1 Parameters

The field is F_P where P is in hexadecimal

P = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF

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

a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC

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

$G = (G_X, G_Y)$

G_X = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296

G_Y = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551

F.11.2.2 Signature key and verification key

X = 5202A3D8 ACAF6909 D12C9A77 4CD886F9 FBA61137 FFD3E8E7 6AED363F B47AC492

$Y = (Y_X, Y_Y)$

Y_X = 9B58B883 23C52D10 80AA525C 89E8E12C 6F40FCB0 14640FA8 8081ED9E 9352DE7

Y_Y = 5CCBBD18 95385162 38B0B0B2 8ACB5F0B 5E27217C 3A987242 1219DE0A EEBF1080

F.11.2.3 Per message data

M = ASCII form of "abc" = 616263

K = DE7E0E5E 663F2418 3414B7C7 2F24546B 81E9E5F4 10BEBF26 F3CA5FA8 2F5192C8

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

Π_X = 847CE3CD 474FEC19 722AA9BA 81AFBF34 7EE2D70E D067413F 1F716783 27A758CA

Π_Y = DBFAD4AF 8C1D93AB 9C16467E 96BD11B5 33643AA6 63498D8F 95919C6C A1AD91FC

$FE2BS(\Pi_X)||FE2BS(\Pi_Y)$ = 847CE3CD 474FEC19 722AA9BA 81AFBF34 7EE2D70E D067413F 1F716783 27A758CA DBFAD4AF 8C1D93AB 9C16467E 96BD11B5 33643AA6 63498D8F 95919C6C A1AD91FC

$R = h(FE2BS(\Pi_X)||FE2BS(\Pi_Y)||M)$ = 5A79A0AA 9B241E38
1A594B22 0554D096 A5F09FA6 28AD9A33 C3CE4393 ADE1DEF7

F.11.2.4 Signature

R = 5A79A0AA 9B241E38 1A594B22 0554D096 A5F09FA6 28AD9A33
C3CE4393 ADE1DEF7

S = 5C0EB78B 67A513C3 E53B2619 F96855E2 91D5141C 7CD0915E
1D04B347 457C9601

F.11.2.5 Verification

$\Pi' = (\Pi'_X, \Pi'_Y)$

Π'_X = 847CE3CD 474FEC19 722AA9BA 81AFBF34 7EE2D70E D067413F
1F716783 27A758CA

Π'_Y = DBFAD4AF 8C1D93AB 9C16467E 96BD11B5 33643AA6 63498D8F
95919C6C A1AD91FC

$FE2BS(\Pi'_X)||FE2BS(\Pi'_Y)$ = 847CE3CD 474FEC19 722AA9BA 81AFBF34
7EE2D70E D067413F 1F716783 27A758CA DBFAD4AF 8C1D93AB
9C16467E 96BD11B5 33643AA6 63498D8F 95919C6C A1AD91FC

$R' = h(FE2BS(\Pi'_X)||FE2BS(\Pi'_Y)||M)$ = 5A79A0AA 9B241E38 1A594B22
0554D096 A5F09FA6 28AD9A33 C3CE4393 ADE1DEF7

F.11.3 Field F_P, 384-bit Prime P

F.11.3.1 Parameters

The field is F_P where P is in hexadecimal

P = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFF

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

a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFC

b = B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112
0314088F 5013875A C656398D 8A2ED19D 2A85C8ED D3EC2AEF

$G = (G_X, G_Y)$

G_X = AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98
59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7

G_Y = 3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C
E9DA3113 B5F0B8C0 0A60B1CE 1D7E819D 7A431D7C 90EA0E5F

q = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973