Standards for Efficient Cryptography

SEC X.1: Supplemental Document for Odd Characteristic Extension Fields

Nippon Telephone and Telegraph Corporation

Contact: Kazumaro Aoki, Tetsutaro Kobayashi, and Akira Nagai
(publickey@lab.ntt.co.jp)

Working Draft
March 2, 2009
Version 0.6

Contents

Update History

Version 0.6 The followings are updated:

- Base document version is replaced from SEC1 Version 1.8 to Version 1.9.
- Add ASN.1 representations.
- Correct editorial mistakes.

Version 0.5 First public release.

1 Introduction

This document is intended to make the odd characteristic extension fields available for elliptic curve cryptography defined in SEC 1. The most of specification in SEC 1 can be used for the odd characteristic extension fields as it is, but some specification should slightly be modified for them. The following sections describe the modifications for the odd characteristic field from SEC 1. Note that the sections that can be used for the odd characteristic extension field as it is are omitted in this document.

2 Mathematical Foundations

2.1 Finite Fields

SEC 1 only defines the finite field $\mathbb{F}_q$, where q is a prime or a power of 2. Here, the odd characteristic extension fields $\mathbb{F}_{p^m}$ are defined, where p is odd prime greater than 3 and $m \geq 2$.

Mathematically speaking, odd characteristic means $p \geq 3$, and $p = 3$ is very special for elliptic curve, and $p = 3$ is not used in this document. $p = 3$ is excluded in the definition of the odd characteristic extension fields.

2.1.X The Finite Field $\mathbb{F}_{p^m}$

The finite field $\mathbb{F}_{p^m}$ is the characteristic p finite field containing p^m elements. This section describes the case that p is prime and $m \geq 2$, and such a field is called odd characteristic extension field. Although there is only one characteristic p finite field $\mathbb{F}_{p^m}$ for each power p^m of p with $m \geq 2$, there are many different ways to represent the elements of $\mathbb{F}_{p^m}$.

Here the elements of $\mathbb{F}_{p^m}$ should be represented using the polynomial basis, that is, the set of polynomials of degree $m-1$ or less with $\mathbb{F}_p$-coefficients:

$$\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 : a_i \in \mathbb{F}_p\}$$

with addition and multiplication defined in terms of an irreducible monic polynomial $f(x)$ of degree m with $\mathbb{F}_p$-coefficient, known as the reduction polynomial, as follows:

- Addition: If $a = a_{m-1}x^{m-1} + \cdots + a_0$, $b = b_{m-1}x^{m-1} + \cdots + b_0 \in \mathbb{F}_{p^m}$, then $a + b = r$ in $\mathbb{F}_{p^m}$, where $r = r_{m-1}x^{m-1} + \cdots + r_0$ with $r_i \equiv a_i + b_i \pmod{p}$.

- Multiplication: If $a = a_{m-1}x^{m-1} + \cdots + a_0$, $b = b_{m-1}x^{m-1} + \cdots + b_0 \in \mathbb{F}_{p^m}$, then $ab = s$ in $\mathbb{F}_{p^m}$, where $s = s_{m-1}x^{m-1} + \cdots + s_0$ is the remainder when the polynomial ab is divided by $f(x)$ with all coefficients arithmetic performed modulo p.

Addition and multiplication in $\mathbb{F}_{p^m}$ can be calculated efficiently using standard algorithms for ordinary integer and polynomial arithmetic. In this representation of $\mathbb{F}_{p^m}$, the additive identity or zero element is the polynomial 0, and the multiplicative identity is the polynomial 1.

Again it is convenient to define subtraction and division of field elements. To do so, the additive inverse (or negative) and multiplicative inverse of a field element must be described:

- Additive inverse: If $a \in \mathbb{F}_{p^m}$, then the additive inverse $(-a)$ of a in $\mathbb{F}_{p^m}$ is the unique solution to the equation $a + x = 0$ in $\mathbb{F}_{p^m}$.
- Multiplicative inverse: If $a \in \mathbb{F}_{p^m}$, $a \neq 0$, then the multiplicative inverse a^{-1} of a in $\mathbb{F}_{p^m}$ is the unique solution to the equation $ax = 1$ in $\mathbb{F}_{p^m}$.

Additive inverse and multiplicative inverses in $\mathbb{F}_{p^m}$ can be calculated efficiently. Multiplicative inverse can be calculated using the polynomial version of the extended Euclidean algorithm. Division and subtraction are defined in terms of additive and multiplicative inverses: $a - b$ in $\mathbb{F}_{p^m}$ is $a + (-b)$ in $\mathbb{F}_{p^m}$ and a/b in $\mathbb{F}_{p^m}$ is $a(b^{-1})$ in $\mathbb{F}_{p^m}$.

Here the odd characteristic extension fields $\mathbb{F}_{p^m}$ used should have:

$$(p, m) \in \{(2^{61} - 1, 5), (2^{61} - 1, 7), (2^{61} - 1, 11)\}$$

and addition and multiplication in $\mathbb{F}_{p^m}$ should be performed using one of the irreducible monic polynomials of degree m in Table 1. As before this restriction is designed to facilitate interoperability while enabling implementers to deploy efficient implementations capable of meeting common security requirements.

Field	Reduction Polynomial(s)
$\mathbb{F}_{(2^{61}-1)^5}$	$f(x) = x^5 - 3$
$\mathbb{F}_{(2^{61}-1)^7}$	$f(x) = x^7 - 3$
$\mathbb{F}_{(2^{61}-1)^{11}}$	$f(x) = x^{11} - 3$

Table 1: Representation of $\mathbb{F}_{p^m}$

The rule used to pick acceptable (p, m)'s was that $\mathbb{F}_{p^m}$ becomes an optimal extension field (OEF). p of an OEF should slightly be less than the power of 2, and the power should also slightly be less than word size in bits. Moreover, (p, m) was chosen such that an irreducible monic binomial with degree m on $\mathbb{F}_p$ exists. $c \in \mathbb{F}_p$ in the irreducible monic binomial $x^m - c$ chosen that as small in non-negative integer as possible. Composite m was avoided to align this specification to address concerns expressed by some experts about the security of elliptic curves defined over $\mathbb{F}_{p^m}$ with m composite.

2.2 Elliptic Curves

2.2.X Elliptic Curves over $\mathbb{F}_{p^m}$

This section is the same as Section 2.2.1 "Elliptic Curves over $\mathbb{F}_p$" in SEC 1, except for the followings:

- All expressions "$\cdots \equiv \cdots \pmod{p}$" are replaced with "$\cdots = \cdots$ in $\mathbb{F}_{p^m}$".
- All prime field notations $\mathbb{F}_p$ are replaced with $\mathbb{F}_{p^m}$.

Note that the case of $p = 3$ is not considered, because Section 2.1.X does not accept this parameter.

2.3 Data Types and Conversions

2.3.3 Elliptic-Curve-Point-to-Octet-String-Conversion

Add the following step in Step 2.2 in **Actions**:
2.2.3. If $q = p^m$ is a power of odd prime, set $\tilde{y}_P = 0$ if $y = 0$, otherwise set $\tilde{y}_P = y_i \pmod 2$, where $y = y_{m-1}x^{m-1} + \cdots + y_1 + y_0$, and i is the smallest integer such that $y_i \neq 0$.

2.3.4 Octet-String-to-Elliptic-Curve-Point Conversion

Add the following step in Step 2.4 in **Actions**:
2.4.4. If $q = p^m$ is a power of odd prime, compute the field element $\alpha = x_P^3 + ax_P + b$ in $\mathbb{F}_{p^m}$, and compute a square root β of α in $\mathbb{F}_{p^m}$. Output "invalid" and stop if there are no square roots of α in $\mathbb{F}_{p^m}$. Otherwise set $y_P = 0$ if $\beta = 0$, otherwise set $y_P = \beta$ if $\beta_i \equiv \tilde{y}_P \pmod 2$, and set $y_P = -\beta$ if $\beta_i \not\equiv \tilde{y}_P \pmod 2$, where $\beta = \beta_{m-1}x^{m-1} + \cdots + \beta_1 x + \beta_0$, and i is the smallest integer such that $\beta_i \neq 0$.

2.3.5 Field-Element-to-Octet-String Conversion

Add the following step in **Actions**:
3. If $q = p^m$ is a power of odd prime, convert a to M as follows:

3.1. Convert a to x using the conversion routine specified in Section 2.3.9 (with a as input).

3.2. Convert x to M using the conversion routine specified in Section 2.3.7 (with a and $mlen$ as inputs).

3.3. Output M.

2.3.6 Octet-String-to-Field-Element Conversion

Add the following step in **Actions**:
3. If $q = p^m$ is a power of odd prime, then a needs to be a polynomial of degree $m-1$ or less with $\mathbb{F}_p$-coefficient. Convert M to a as follows:

3.1. Convert M to an integer x using the conversion routine specified in Section 2.3.8.

3.2. Output "invalid" and stop if x does not lie in the interval $[0, p^m - 1]$.

3.3. View a_i as an integer in the range $[0, p-1]$ and set:

$$x = \sum_{i=0}^{m-1} p^i a_i.$$

3.4. Set the field element a to be $a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, and output a.

2.3.9 Field-Element-to-Integer Conversion

Add the following step in **Actions**:
3. If $q = p^m$ is a power of odd prime, then a must be a polynomial of degree $m-1$ or less with $\mathbb{F}_p$-coefficient, — i.e. $a = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$. Set:

$$x = \sum_{i=0}^{m-1} p^i a_i$$

Output x.

3 Cryptographic Components

3.1 Elliptic Curve Domain Parameters

Following sections often consider only two types of elliptic curve domain parameters. Whenever "two types" are found, consider that the types should include the type for odd characteristic extension field.

3.1.X Elliptic Curves Domain Parameters over $\mathbb{F}_{p^m}$

Elliptic curve domain parameters over $\mathbb{F}_{p^m}$ are an octuple:

$$T = (p, m, f(x), a, b, G, n, h)$$

consisting of an integer p and m specifying the finite field $\mathbb{F}_{p^m}$, two elements $a, b \in \mathbb{F}_{p^m}$ specifying the elliptic curve $E(\mathbb{F}_{p^m})$ defined by the equation:

$$E : y^2 = x^3 + ax + b \text{ in } \mathbb{F}_{p^m},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_{p^m})$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_{p^m})/n$.

Elliptic curve domain parameters over $\mathbb{F}_{p^m}$ precisely specify an elliptic curve and base point. This is necessary to precisely define public-key cryptographic schemes based on ECC.

If the elliptic curve domain parameters T are verifiably random, then they should be accompanied by the seed value S from which they are derived. Section 3.1.X.1 describes how to generate elliptic curve domain parameters over $\mathbb{F}_{p^m}$, and Section 3.1.X.2 describes how to validate elliptic curve domain parameters over $\mathbb{F}_{p^m}$.

3.1.X.1 Elliptic Curve Domain Parameters over $\mathbb{F}_{p^m}$ Generation Primitive

Elliptic curve domain parameters over $\mathbb{F}_{p^m}$ should be generated as follows:

Input: The approximate security level in bits required from the elliptic curve domain parameters — this must be an integer $t \in \{80, 112, 128, 192, 256\}$. Optionally, a seed value S.

Output: Elliptic curve domain parameters over $\mathbb{F}_{p^m}$:

$$T = (p, m, f(x), a, b, G, n, h)$$

such that taking logarithms on the associated elliptic curve requires approximately 2^t operations.

Actions: Generate elliptic curve domain parameters over $\mathbb{F}_{p^m}$ as follows:

1. Select $(p, m) \in \{(2^{61} - 1, 5), (2^{61} - 1, 7), (2^{61} - 1, 11)\}$ to determine the finite field $\mathbb{F}_{p^m}$ such that $2^{2t} < p^m$ for a random curve or $2^{2t} < p^{m-1}$ for a Koblitz curve.
2. Select an irreducible monic polynomial $f(x)$ of degree m from Table 1 in Section 2.1.X to determine the representation of $\mathbb{F}_{p^m}$.
3. Select elements $a, b \in \mathbb{F}_{p^m}$ to determine the elliptic curve $E(\mathbb{F}_{p^m})$ defined by the equation:

   $$E : y^2 = x^3 + ax + b \text{ in } \mathbb{F}_{p^m},$$

   a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_{p^m})$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_{p^m})/n$, subject to the following constraints:

- $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_{p^m}$.
- $\#E(\mathbb{F}_{p^m}) \neq p^m$.
- $p^B \not\equiv 1 \pmod{n}$ for any $1 \leq B < 100m$.
- $h \leq p2^{t/8}$.
- $n-1$ and $n+1$ should each have a large prime factor r, which is in the sense that $\log_n(r) > \frac{19}{20}$.

If seed S is provided, then the coefficient pair (a, b), or the point G should be derived from S, or both See Section 3.1.3.

4. Output $T = (p, m, f(x), a, b, G, n, h)$.

This primitive allows any of the known curve selection methods to be used — for example the methods based on complex multiplication and the methods based on general point counting algorithms. However to foster interoperability it is strongly recommended that implementers use one of the elliptic curve domain parameters over $\mathbb{F}_{p^m}$ specified in SEC X.2. See Appendix B for further discussion.

3.1.X.2 Validation of Elliptic Curve Domain Parameters over $\mathbb{F}_{p^m}$

This section is the same as Section 3.1.1.2 "Validation of Elliptic Curve Domain Parameters over $\mathbb{F}_p$" in SEC 1, except for the followings:

- All prime field notation $\mathbb{F}_p$ is replaced with $\mathbb{F}_{p^m}$.
- "Section 3.1.1.2.1" is replaced with "Section 3.1.X.2.1".
- "Section 3.1.1.1" is replaced with "Section 3.1.X.1".

3.1.X.2.1 Elliptic Curve Domain Parameters over $\mathbb{F}_{p^m}$ Validation Primitive

The elliptic curve domain parameters over $\mathbb{F}_{p^m}$ validation primitive should be used to check that elliptic curve domain parameters over $\mathbb{F}_{p^m}$ are valid as follows:

Input: Elliptic curve domain parameters over $\mathbb{F}_{p^m}$:

$$T = (p, m, f(x), a, b, G, n, h)$$

along with an integer $t \in \{80, 112, 128, 192, 256\}$ which is the approximate security level in bits required from the elliptic curve domain parameters.

Output: An indication of whether the elliptic curve domain parameters are valid or not — either "valid" or "invalid".

Actions: Validate the elliptic curve domain parameters over $\mathbb{F}_{p^m}$ as follows:

1. Check that (p, m) is an integer in the set $\{(2^{61}-1, 5), (2^{61}-1, 7), (2^{61}-1, 11)\}$ such that $2^{2t} < p^m$ for a random curve or $2^{2t} < p^{m-1}$ for a Koblitz curve.
2. Check that $f(x)$ is an irreducible monic polynomial of degree m with $\mathbb{F}_p$-coefficient which is listed in Table 1 in Section 2.1.X.
3. Check that a, b, x_G, and y_G are in $\mathbb{F}_{p^m}$.
4. Check that $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_{p^m}$.
5. Check that $y_G^2 = x_G^3 + ax_G + b$ in $\mathbb{F}_{p^m}$.
6. Check that n is prime.
7. Check that $h \leq p2^{t/8}$, and that $h = \lfloor(\sqrt{p^m} + 1)^2/n\rfloor$.
8. Check that $nG = \mathcal{O}$.
9. Check that $p^B \not\equiv 1 \pmod{n}$ for any $1 \leq B < 100m$, and that $nh \neq p^m$.
10. If any of the checks fail, output "invalid", otherwise output "valid".

Step 9 above excludes the known weak classes of curves which are susceptible to either the Menezes-Okamoto-Vanstone attack, or the Frey-Rück attack, or Semaev-Smart-Satoh-Araki attack. See Appendix B in SEC 1 for further discussion.

If the elliptic curve domain parameters have generated verifiably at random as described in Section 3.1.3, it may also be checked that a and b have been correctly derived from the seed, and it may also checked that G has been correctly derived from S.

3.1.3 Verifiably Random Curves and Base Point Generators

The subsections of this section include several unclear descriptions. The following is the modification proposal. When modifying the procedures, the following problems are considered:

- *Hash* is assumed as one defined in Section 3.5, that is, *Hash* inputs an octet string and also outputs an octet string, not a bit string.
- Unlike other procedures, one of inputs, "seed", is a bit string, not an octet string.

3.1.3.1 Curve Selection

For generating Koblitz curve over $\mathbb{F}_{p^m}$, the input of field size q is chosen as p instead of p^m, otherwise choose $q = p^m$.

Input: A "seed" octet string S of length g, field size q, hash function *Hash* of output length t, and field element $a \in \mathbb{F}_q$.

Output: A field element $b \in \mathbb{F}_q$ or "failure".

Actions: Generate the element b as follows:

1. If $a = 0$, then output "failure" and stop.
2. Let $u = \lceil \log_2 q \rceil$.
3. Let $s = \lfloor (u-1)/t \rfloor$.
4. Let $k = u - st$ if q is even, and let $k = u - st - 1$ if q is odd.
5. Convert S to an integer s_0 using the conversion routine specified in Section 2.3.8 (with S as input).
6. For j from 0 to s, do the following:
   6.1 Let $s_j = s_0 + j \bmod 2^{8g}$.
   6.2 Let S_j be the integer s_j converted to an octet string of length g octets using the conversion routine specified in Section 2.3.7 (with S_j and g as inputs).
   6.3 Let $H_j = \mathit{Hash}(S_j)$.
   6.4 Convert H_j to an integer e_j using the conversion routine specified in Section 2.3.8 (with H_j and t as inputs).
7. Let $e = e_0 2^{ts} + e_1 2^{t(s-1)} + \cdots + e_s \bmod 2^{k+st}$.
8. Convert e to an octet string x using the conversion routine specified in Section 2.3.7 (with e and $\lceil u/8 \rceil$ as inputs).
9. Convert x to a field element $r \in \mathbb{F}_q$ using the conversion routine specified in Section 2.3.6 (with the indication of the field $\mathbb{F}_q$ and x as inputs).
10. If q is even, then do as follows:
    10.1 If $r = 0$, then output "failure" and stop.
    10.2 If $r \neq 0$, then output $b = r \in \mathbb{F}_q$ and stop.
11. If q is odd, then do as follows:
    11.1 If $r = 0$, then output "failure" and stop.
    11.2 If $4r + 27 = 0$ in $\mathbb{F}_q$, then output "failure" and stop.
    11.3 If a^3/r does not have a square root in $\mathbb{F}_q$, then output "failure" and stop.
    11.4 Otherwise, choose one of $\pm\sqrt{a^3/r} \in \mathbb{F}_q$ as b and output b and stop.

3.1.3.2 Point Selection

Input: A "seed" octet string S of length g, field size q, hash function *Hash* of output length t, and elliptic curve parameters a and b, and elliptic curve cofactor h.

Output: An elliptic curve point G.

Actions: Generate an elliptic curve point G as follows:

1. Let $A = 4261736520706F696E74_{16}$, which is the octet string associated with the ASCII representation of the text string "Base point".
2. Let $B = 01_{16}$, an octet string of length 1.
3. Let an integer $c = 1$.
4. Convert integer c to an octet string C of length $1 + \lfloor \log_{256}(c) \rfloor$ using the conversion routine specified in Section 2.3.7 (with c and $1 + \lfloor \log_{256}(c) \rfloor$ as inputs).
5. Let $H = \mathit{Hash}(A \| B \| C \| S)$.
6. Convert H to an integer e using the conversion routine specified in Section 2.3.8 (with H as input).
7. Let $s = e \bmod 2q$.
8. Let $u = s \bmod q$ and $\tilde{y}_R = \lfloor s/q \rfloor$.
9. Convert integer u to an octet string x using the conversion routine specified in Section 2.3.7 (with u and $1 + \lfloor \log_{256} q \rfloor$ as inputs).
10. Convert octet string x to a field element $x_R \in \mathbb{F}_q$ using the conversion routine specified in Section 2.3.6 (with the indication of the field $\mathbb{F}_q$ and x as inputs).
11. Recover a y-coordinate from the compressed point information $(x_R, \tilde{y}_R)$, as appropriate to the elliptic curve followed by Step 2.4 in Section 2.3.4.
12. If there is no valid y_R, then increment c and go back to Step 4.
13. Let $R = (x_R, y_R)$.
14. Compute $G = hR$.
15. Output G.

3.2 Elliptic Curve Key Pairs

Modification request: Please do not use the specific descriptions regarding $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ such as "$T = (p, a, b, G, n, h)$ or $(m, f(x), a, b, G, n, h)$" in the schemes including ECDSA for the following sections.

A Glossary

A.1 Terms

Add the following term(s):

odd characteristic extension field A finite field containing p^m elements, where $p > 3$ is an odd prime number and $m \geq 2$ is an integer.

Modify the following term(s):

reduction polynomial The irreducible monic polynomial $f(x)$ of degree m with $\mathbb{F}_2$- or $\mathbb{F}_p$-coefficient that is used to determine a representation of $\mathbb{F}_{2^m}$ or $\mathbb{F}_{p^m}$.

A.2 Acronyms

Add the following acronym(s):

OEF optimal extension field

A.3 Notation

Modify the following notation(s):

$\mathbb{F}_q$ The finite field containing q elements. In this document attention is restricted to the cases that q is an odd prime number p or a power of 2 (2^m) or a power of an odd prime p (p^m).

B ASN.1

```
--########################################################################
-- finite field, group, and elliptic curve representations
Group ::= CHOICE {
groupOid OBJECT IDENTIFIER,
groupHashId OCTET STRING, -- defined in RFC2528
groupParameters GroupParameters
}
GroupParameters ::= CHOICE {
explicitFiniteFieldSubgroup
[0] ExplicitFiniteFieldSubgroupParameters,
ellipticCurveSubgroup
[1] EllipticCurveSubgroupParameters
}
```

```
ExplicitFiniteFieldSubgroupParameters ::= SEQUENCE {
fieldID FieldID {{FieldTypes}},
generator FieldElement,
subgroupOrder INTEGER,
subgroupIndex INTEGER
}
FIELD-ID ::= TYPE-IDENTIFIER
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
fieldType FIELD-ID.&id({IOSet}),
parameters FIELD-ID.&Type({IOSet}{@fieldType}) OPTIONAL
}
FieldTypes FIELD-ID ::= {
{ Prime-p IDENTIFIED BY prime-field } |
{ Characteristic-two IDENTIFIED BY characteristic-two-field }|
{ Odd-characteristic IDENTIFIED BY id-ft-odd-characteristic },
... -- expect additional field types
}
-- prime fieds
Prime-p ::= INTEGER
-- characteristic two fields
CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
-- when basis is gnBasis then the basis shall be an optimal
-- normal basis of Type T where T is determined as follows:
-- if an ONB of Type 2 exists for the given value of m then
-- T shall be 2, otherwise if an ONB of Type 1 exists for the
-- given value of m then T shall be 1, otherwise T shall be
-- the least value for which an ONB of Type T exists for the
-- given value of m
-- when basis is gnBasis then m shall not be divisible by 8
-- note: the above rule is from ANSI X9.62
-- note: for the given m and T the ONB is unique
Characteristic-two ::= SEQUENCE {
m INTEGER,-- extension degree
basis CHARACTERISTIC-TWO.&id({BasisTypes}),
parameters CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}
BasisTypes CHARACTERISTIC-TWO ::= {
{ NULL IDENTIFIED BY gnBasis } |
{ Trinomial IDENTIFIED BY tpBasis } |
{ Pentanomial IDENTIFIED BY ppBasis } |
{ CharTwoPolynomial IDENTIFIED BY charTwoPolynomialBasis },
... -- expect additional basis types
}
```

```
Trinomial ::= INTEGER
Pentanomial ::= SEQUENCE {
k1 INTEGER,
k2 INTEGER,
k3 INTEGER
}
-- characteric two general irreducible polynomial representation
-- the irreducible polymial
-- a(n)*x^n + a(n-1)*x^(n-1) + ... + a(1)*x + a(0)
-- is encoded in the bit string with a(n) in the first bit, the
-- following coefficients in the following bit positions and a(0)
-- in the last bit of the bit string (one could omit a(n) and a(0)
-- but it may be simpler and less error-prone to leave them in
-- the encoding)
-- the degree of the polynomial is to be inferred from the length
-- of the bit string
CharTwoPolynomial ::= BIT STRING
-- odd characteristic extension fields
ODD-CHARACTERISTIC ::= TYPE-IDENTIFIER
Odd-characteristic ::= SEQUENCE {
characteristic INTEGER(3..MAX),
degree INTEGER(2..MAX),
basis ODD-CHARACTERISTIC.&id({OddCharBasisTypes}),
parameters ODD-CHARACTERISTIC.&Type({OddCharBasisTypes}{@basis})
}
OddCharBasisTypes ODD-CHARACTERISTIC ::= {
{ OddCharPolynomial IDENTIFIED BY oddCharPolynomialBasis },
... -- expect additional basis types
}
-- the monic irreducible polynomial is encoded as follows
-- the leading coefficient is ignored
-- the remaining coefficients define an element of the finite field
-- which is encoded in an octet string using FE2OSP
OddCharPolynomial ::= FieldElement
EllipticCurveSubgroupParameters ::= SEQUENCE {
version INTEGER { ecpVer1(1) } (ecpVer1),
fieldID FieldID {{ FieldTypes }},
curve Curve,
generator ECPoint, -- Base point G
subgroupOrder INTEGER, -- Order mu of the base point
subgroupIndex INTEGER, -- The integer nu = #E(F)/mu
...
}
```

```
Curve ::= SEQUENCE {
aCoeff FieldElement,
bCoeff FieldElement,
seed BIT STRING OPTIONAL
}
```

∗ Possible Corrections or Comments on SEC 1 (Draft Version 1.9)

page 1, Section 1, 1st line: "sectiongives" should be "section gives".

page 14, Section 2.3.8, 1st line: "range $[1, 256]$" should be "range $[0, 255]$". Note that Version 1.0 does not contain this problem.

page 14, Section 2.3.9, 3rd line: The formal procedure does not convert the binary polynomial to an octet string.

page 14, Section 2.3.9, Step 2 in Actions: The degree of a may be less than $m - 1$.

page 16, Section 3.1.1.1, last line in Step 2 in Actions: "§" is used instead of "Section".

page 18, Section 3.1.2.1, Step 8 in Actions: "$n \neq p$" should be replaced with "$nh \neq p$".

page 19, Section 3.1.2.1, last line in Step 3 in Actions: "§" is used instead of "Section".

page 20, Section 3.1.2.2, last line: The description for CA is omitted while Section 3.1.1.2 writes.

page 21, Section 3.1.3: This section only generates an elliptic curve or a base point. Sections 3.1.1.2.1 and 3.1.2.2.1 certainly describe the use of seed S. However, this section does not express the generated curves (or base point) is valid for the use of secure cryptosystem.

page 21, Section 3.1.3, 2nd line: A verification procedure such as Appendix 6.5 in FIPS186-2 (+ Change Notice) cannot be found. (See Appendix †.1 for the problem in FIPS186-2.)

page 21, Section 3.1.3, 2nd line: "ANS" may be replaced with "ANSI".

page 21, Section 3.1.3, 2nd paragraph, 4th line: "based point" should be replaced with "base point".

page 21, Section 3.1.3.1: The following problems are identified:

- Section 3.5 seems to define *Hash* outputs octet string, but t seems a number of bits.
- "is q is odd" in Step 3 should be replaced with "if q is odd".
- Which conversion procedure is used is not written in Steps 4 and 7.

- Step 7 may output "invalid".
- Step 9.1 does not confirm the case $r = 0$.
- Lack of period in Step 9.1.
- In Step 9.2, "in $\mathbb{F}_q$" should be added for $4r + 27 = 0$.
- How to choose one of $\sqrt{a^3/r}$ in Step 9.4?

page 22, Section 3.1.3.2: The following problems are identified:

- The output of the procedure includes "failure". However, the procedure outputs "failure" when c is almost larger than the input bound of *Hash*. It is very hardly occurs.
- t is used for the output length of *Hash* and Step 7.
- If the output length of *Hash* is small, u is not uniform when q is odd. Moreover, z is always 0 in Step 10.
- The compressed point information (x, z) in Step 10 is not defined before.

page 22-, Section 3.2-: The following sections only considers for the prime field and the characteristic two field. It seems very easy to modify that the descriptions for prime fields considers for both prime field odd characteristic extension fields. Could you include this comments?

page 29-30, Section 3.5: The hash functions written in the section inputs and outputs octet strings, but hash function *Hash* in **Actions** inputs and outputs bit strings.

page 72-73, Section B.2.1: Add the comments regarding the following problem. Section 3.1.3 describes the verifiable parameter generations. Some parameters generated by the procedures described in the section are not chosen uniformly random from acceptable range. This may not be a problem, but the thought why the procedures are okay should be described.

page 96-99, Section B.6: Many "ANS" are found. Actually, the expression both ANS X9.62 and ANSI X9.62 seems acceptable, but the combination seems strange.

† Verifiable Random Curve Generation in Standards

†.1 FIPS

c is generated by SHA-1, and one cannot control the value of c.

- Step 9 of APPENDIX 6.4 (page 63) in FIPS 186-2 (+ Change Notice)
  Step 9 of Section E.5 (page 114) in FIPS 186-3 (March 2006 Draft)

  9. Choose integers $a, b \in GF(p)$ such that

  $$cb^2 \equiv a^3 \pmod{p}$$

  (The simplest choice is $a = c$ and $b = c$. However, one may want to choose differently for performance reasons.)

- Step 7 of APPENDIX 6.5 (page 64) in FIPS 186-2 (+ Change Notice)
  Step 7 of Section E.6 (pp.114–115) in FIPS 186-3 (March 2006 Draft)

  7. Verify that $b^2c \equiv -27 \pmod{p}$.

The verification procedure seems to assume $a = -3$, but the generation procedure accepts the case of $a \neq -3$.

†.2 SEC

r is derived with *Hash*, and one cannot control the value of r. a is one of the input of the generation procedure, similar to the input of hash function. In SEC 1, Draft 1.9:

- Step 9 (page 21) in Section 3.1.3.1:

  9.1 If $a = 0$, then output "failure" and stop
  9.2 If $4r + 27 = 0$, then output "failure" and stop.
  9.3 If a^3/r does not have a square root in $\mathbb{F}_q$, then output "failure" and stop.
  9.4 Otherwise output $b = \sqrt{a^3/r} \in \mathbb{F}_q$ and stop.

- The verification procedure is undefined.

†.3 ANSI

r is derived with *Hash*, and one cannot control the value of r. a is arbitrary chosen. In ANSI X.62-2005:

- Step l in Section A.3.3.1

  1) Solve the equation $b^2r = a^3$ for b in F_q if a solution exists; otherwise, stop and output "Failure" (see NOTE 2).
  2) If $4a^3 + 27b^2 = 0$, then stop and output "Failure" (see NOTE 3).

- Step d in Section A.3.3.2

  If *SEED* is provided and the curve E does not match the curve that would be generated using A.3.3.1, then stop and output "Invalid".

†.4 Other Comments

- The random number generated from hash function is not uniform in the valid range. Uniformly random value seems better.
- To decide a and b in the equation $y^2 = x^3 + ax + b$ has some freedom. The algebraic closure of the curve has the same group structure, however, the order of the elliptic curve over the finite field may vary. It seems better that a and b does not have freedom to choose by the generation.