ECC Brainpool Standard Curves and Curve Generation
v. 1.0
19.10.2005

The OID of this paper is:
`{iso(1) identified-organization(3) teletrust(36) algorithm(3) signature-algorithm(3) ecSign(2) ecStdCurvesAndGeneration(8)}`

1. Introduction.

This paper proposes a set of elliptic curve domain parameters over finite prime fields $GF(p)$ [1]. Although there are several existing proposals for standard elliptic curves (see Section 4), some major issues have still not been addressed:

- The elliptic curves specified do not cover all bit lengths that correspond to the commonly used key lengths for symmetric cryptographic algorithms.
- The choice of the seeds from which the curve parameters have been derived is not motivated leaving an essential part of the security analysis open.
- No proofs are provided that the proposed curves do not belong to those classes of curves for which more efficient cryptanalytic attacks are possible.
- Recent research results justify additional security requirements for elliptic curves (e.g. the class group condition, see Section 3) at least for applications with highest security demands.
- Some of the proposed subgroups have non-trivial cofactor, which demands additional checks by cryptographic applications to prevent small subgroup attacks.

Furthermore, even though there are several proposed sets of elliptic curves for cryptographic applications most of the relevant curves are identical. Thus, there is still a need for a proposal of additional curves suited for cryptographic applications with highest security demands complemented with comprehensive security proofs. The present paper aims at satisfying this need. For example, the 224 bit curve brainpoolP224r1 and the 256 bit curve brainpoolP256r1 described below will be used in the new German machine readable travel documents (MRTDs) that follow ICAO technical reports [ICAO].

It is envisioned to provide additional curves on a regular basis for users who wish to change curve parameters regularly, cf. Annex H2 of [X9.62], paragraph „Elliptic curve domain parameter cryptoperiod considerations“.

Future editions of this paper may also contain elliptic curves over fields of characteristic 2.

Attention is drawn to the fact that some of the mechanisms described in this paper may be subject to patent rights. The identification of such patent rights is beyond the scope of this paper.

In Section 2 a basic background on elliptic curves for cryptographic applications is given. Section 3 specifies the initial technical and security requirements imposed by cryptographic applications. Section 4 summarizes the most important proposals for elliptic curves. In Section 5 the procedures for pseudo-random generation of the parameters are specified. Section 6 gives an initial overview of the class-group condition which is supplemented by Appendix A. Section 7 describes the tests performed during the curve-evaluation. Section 8 describes data formats and the curve naming conventions used in this paper. The proposed curves and the respective validation data are given in Sections 10 and 11.

Appendix B describes the Abtract Syntax Notation One (ASN.1) syntax that can be used to describe elliptic curves. The ASN.1 coding according to X9.62 [X9.62] of all proposed curves is provided in this appendix. In addition Appendix B gives Object Identifiers (OIDs) for all

[1] We choose $\{0,\dots,p-1\}$ as a set of representatives for the elements of $GF(p)$. Note that this choice induces a natural ordering on GF(p).

curves. All data can be accessed through the webpage of the ECC Brainpool [BP] or through the webpage of Teletrust [TTT]. The webpage of the ECC Brainpool also provides additional tools and information, e.g. a binary coding of the ASN.1 data.

2. Basic properties of elliptic curves.

For $p > 3$ every elliptic curve over the finite field $GF(p)$ can be described (see Remark 1 below) as the set of solutions (over an algebraic closure of $GF(p)$) of an equation

$$E : y^2 = x^3 + Ax + B \mod p,$$

together with a „point at infinity" **O**. This set forms an abelian group with neutral element **O**, the addition law can for example be found in [Sil]. Its subgroup $E(GF(p))$ of points defined over $GF(p)$ has order $\# E(GF(p))$ with

$$|\# E(GF(p)) - (p+1)| \le 2 \cdot \sqrt{p}$$

by the theorem of Hasse-Weil. For cryptographic applications one usually chooses a cyclic subgroup of $E(GF(p))$ of prime order *q* with small cofactor $\# E(GF(p))/q$.

Details on elliptic curves may be found in the book of Silverman [Sil]. The use of elliptic curves in cryptography is explained in [BSS], [CF] or [HMV].

Let P_0 be an element of prime order *q* of $E(GF(p))$ and Q be contained in the cyclic subgroup generated by P_0. For the security of elliptic curve based cryptographic mechanisms the hardness of the *Elliptic Curve Discrete Logarithm Problem* (ECDLP) in the chosen cyclic subgroup is necessary. (But not always sufficient. Related problems are for example the *Elliptic Curve Diffie-Hellman* or the *Elliptic Curve Decision Diffie-Hellman* problem.)

The ECDLP is the problem of finding a number *k* between 1 and *q* fulfilling $Q = k \cdot P_0$. For suitably chosen elliptic curves the best presently known algorithm for solving the ECDLP is *Pollard's Rho* method: Its expected running time is approximately $\sqrt{pq/2}$. Pollard's Rho method is parallelizable [HMV] with a speedup that is linear in the number of processors employed. It should also be noted that it is easier to solve multiple ECDLPs on one curve than on different curves [KS], [HMCD].

A comparison of the expected running time of Pollard's Rho algorithm with the running time of factoring algorithms or of algorithms for the Discrete Logarithm Problem (DLP) in finite fields shows that elliptic curve systems can offer higher security with shorter key lengths than other commonly used public key systems (i.e. RSA or DLP-based systems).

In practice cryptographic systems usually are hybrid systems: An asymmetric algorithm is used as key-management algorithm for a symmetric encryption algorithm and for generating digital signatures. Therefore the symmetric and the asymmetric algorithm should offer a comparable level of security. The following comparison of security levels provided by symmetric algorithms, elliptic curves and RSA for various key lengths was published by NIST [NIST] (the ECC key length refers to the bit length of *q*, the RSA key length to the bit length of the modulus)

Symmetric key length	80	112	128	160	256
ECC, bit length of q	160	224	256	320	512
RSA modulus bit length	1024	2048	3072	7680	15360

Recently, cryptographic applications based on "insecure" elliptic curves, e.g. for three party key-agreement protocols and for identity based cryptography, are discussed. These applications are not within the scope of this paper. We concentrate on curves fulfilling the security criteria given in Section 4 below, and which are suitable for elliptic curve based signature and key-management algorithms.

Remark 1. The definition $E: y^2 = x^3 + Ax + B \mod p$ is called „Short Weierstrass Form“. There are other descriptions of elliptic curves that allow for more efficient implementations, but the choices made below exclude most curves that can be represented in these other forms.

3. Requirements on elliptic curves for cryptographic applications.

3.1 Technical Requirements

Commercial demands and experience with existing implementations lead to the following technical requirements for the proposed curves. One should note that all choices made below can influence the determination of an optimal implementation that is not susceptible to side-channel attacks.

1. For each of the bit lengths 160, 192, 224, 256, 320, 384 and 512 one curve shall be proposed. This requirement follows from the need for curves providing different levels of security which are appropriate for the underlying symmetric algorithms. Most of the previous proposals specify a 521-bit curve instead of a 512-bit curve.
2. The prime number *p* shall be congruent $3 \mod 4$. This requirement allows efficient point compression: One method for the transmission of curve points $P = (x, y)$ is to transmit only x and the least significant bit *LSB(y)* of *y*. Using the curve equation and the fact that if $p = 3 \mod 4$ then $(y^2)^{(p+1)/4} = y^{(p-1)/2} \cdot y \mod p$, which is either *y* or *–y* by Fermat's little theorem, *y* can be computed very efficiently.
3. The curves shall be $GF(p)$-isomorphic to a “cryptographically good curve” (i. e. a curve that meets all security requirements defined in section 3.2) with $A = -3 \mod p$. This property allows to use the arithmetical advantages of curves with $A = -3 \mod p$ as shown by Brier and Joyce [BJ]. The requirement is fulfilled by a quadratic twist E_1 of the given curve E with a square in $GF(p)$. If $-3 = AZ^4 \mod p$ is solvable, then E and
   $E_1 : y^2 = x^3 + Z^4 \cdot A \cdot x + Z^6 \cdot B = x^3 - 3x + Z^6 \cdot B \mod p$
   are *GF(p)*-isomorphic via the isomorphism $F(x,y) := (x \cdot Z^2, y \cdot Z^3)$. Especially $\#E(GF(p)) = \#E_1(GF(p))$ and, most importantly, E and E_1 have the same algebraic structure and hence, offer the same level of security. Approximately, half of the isomorphism classes of elliptic curves over $GF(p)$ with $p = 3 \mod 4$ contain a curve with $A = -3 \mod p$.
4. The prime *p* must not be of a special form in order to avoid patented fast arithmetic on the base field; our generation method for primes is described in Section 6 below. The need for such curves over pseudo-randomly chosen fields $GF(p)$ has already been foreseen by the Standards for Efficient Cryptography Group (SECG), see Section 4.
5. $\#E(GF(p)) < p$. As a consequence of the Hasse-Weil-theorem the number of points $\#E(GF(p))$ may be greater than the characteristic p of the prime field $GF(p)$. In some cases even the bit-length of *#E(GF(p))* can exceed the bit-length of *p*. To avoid overruns in implementations we require that $\#E(GF(p)) < p$. In connection with digital signature schemes some authors propose to use $q > p$ for security reasons, but the attacks described e. g. in [BRS] appear infeasible in a thoroughly designed PKI.
6. *B* shall be a non-square mod *p*. Otherwise the compressed representations of $(0,0)$ and of the curve-point $(0, X)$ with *X* being the square root of *B* that has least significant bit 0 would be identical. As there are implementations of elliptic curves that encode the point at

infinity as (0,0) we try to avoid ambiguities. Note that this condition is stable under quadratic twists as described in Condition 3 above. Condition 6 makes an attack described in [G] impossible. It can therefore also be seen as a security requirement.

7. All proposed curves shall have an OID in order to facilitate imlementations.
8. The curve data shall also be given in ASN.1 syntax in order to facilitate implementations.

3.2 Security Requirements.

Security requirements are requirements motivated by crypto-mathematical attacks and requirements that enhance trust in the recommended curves.

1. **Immunity to attacks using the Weil- or Tatepairing.** Those attacks allow the embedding of the cyclic subgroup of E into the group of units of a degree-l extension $GF(p^l)$ of $GF(p)$, where subexponential attacks on the DLP exist. Here we have $l = \min\{t \mid q \text{ divides } p^t - 1\}$, i.e. l is the order of p mod q. By Fermat's little theorem l divides $q-1$. We compute the exact value of l by factoring $q-1$. Our requirement is that the quotient $(q-1)/l < 100$. That means l is close to the maximal possible value. Detailed information can be found in [BSS].
   Note 1. Over *GF(p)* this requirement excludes supersingular curves, because those are the curves of order $p+1$, and $p+1$ divides $p^2 - 1$.
   Note 2. Therefore the proof of security for 512-bit curves requires the factorisation of a number of up to 512 bit length.
   Note 3. [SEC1, p.17] and [X9.62, Annex A] only require $l \geq 20$. Therefore our security requirement is considerably stronger.
2. **The curves are not trace one curves.** Trace one curves (or anomalous curves) are curves with *#E(GF(p)) = p*. Satoh and Araki [SA], Semaev [Sem] and Smart [Sma] independently proposed efficient solutions to the ECDLP on trace one curves. Note that these curves are also excluded by Requirement 5 of Section 3.1.
3. **The class number of the maximal order of the endomorphism ring of *E* is larger than 10000000.** *E* cannot be lifted to a curve *E'* over an algebraic number field *L* with *End(E)* = *End(E')* unless the degree of *L* over the rationals is larger than the class number of *End(E).* This security condition was first mentioned in [Spa]. Although there are no efficient attacks exploiting a small class number, recent work [JMV] (see Remark 1 in Section 6) and [HR] also may be seen as argument for the class number condition. See Section 6 and Appendix A for more details on class group computations. This condition excludes curves that are generated by the well-known CM-method.
4. **Group order.** The group order *#E(GF(p))* shall be a prime number q in order to counter small-subgroup attacks (cf. [HMV]). Therefore all groups proposed in this paper have cofactor 1. Note that curves with prime group order have no point of order 2 and therefore no point with *y*-coordinate 0.
5. **Verifiably pseudo-random.** The curves shall be generated in a pseudo-random manner using seeds that are generated in a systematic and comprehensive way. Our method of construction is explained in Section 5.
6. **Proof of security.** For all curves a proof should be given that all security requirements are met. Details are explained below.

From [BG] one could derive the requirement that $q-1$ be "almost" prime. As this condition comes from attacks on some elliptic curve based cryptographic mechanisms that do not appear to be practical it is not included in this paper. This decision may be reconsidered for future editions of this paper.

4. Existing proposals.

There are several existing proposals for elliptic curves for cryptographic applications. We mention only the most important contributions, an overview is for example given in Appendix B of [HMV], sample curves are listed in Appendix A of [HMV].

NIST. The Digital Signature Standard [FIPS 186-2] recommends a series of elliptic curves for Federal Government use. All proposed curves over $GF(p)$ have cofactor 1 and the special form $E: y^2 = x^3 - 3x + B \bmod p$. Curves for 192, 224, 256, 384, and 521 (not 512!) bits are provided.

SECG. The SECG has laid out its proposal in [SEC1] and [SEC2]. All fields $GF(p)$ proposed by the SECG have special properties, as stated in [SEC2, p. 3]: *"All the recommended curve domain parameters over GF(p) use special form primes for their field order p. These special form primes facilitate especially efficient implementations [...] Recommended elliptic curve domain parameters over GF(p) which use random primes for their field order p may be added later if commercial demand for such parameters increases."*
The prime numbers, for which curves are given have bit length 112, 128, 160, 192, 224, 256, 384 and 521. It is required that the cofactor is at most 4 [SEC1, p. 17]. The paper recommends curves which are isomorphic to a curve with $A = -3 \bmod p$, as such curves admit efficient implementation [SEC2, p. 4]. These curves coincide with the NIST-curves for the corresponding bit-lengths.

ANSI. The X9.62 standard mentions two sample curves over *GF(p);* one is taken from [FIPS 186-2], the other one has the bit-length 239. The members of the ECC-Brainpool have no identified need for an additional curve with this bit-length. Therefore this paper does not propose a 239-bit curve.

IETF. RFC 2409 and RFC 2412 support the third and fourth Oakley Group. These are elliptic curves over $GF(2^{155})$ and $GF(2^{185})$, respectively.

5. Pseudo-random generation of parameters.

5.1 Generation of prime numbers.

This Section describes the choice of the base fields $GF(p)$ proposed in this paper. The prime generation method is similar to the method given in FIPS PUB 186-2 [FIPS 186-2], Appendix 6.4, and ANSI X9.62 [X9.62], A.3.2. It is a modification of the method given in 8.2.2 (Incremental search) of ISO/IEC 18032 [ISO].
We use the following update function for seeds:

Function update_seed
Input: A 160-bit string *s*.
Let *z* be the integer whose binary expansion is given by the 160-bit string s.
Define the 160-bit string *t* to be the binary expansion of the integer $(z+1) \bmod 2^{160}$.
Output: *t*

In order to generate an *L*-bit prime *p* define

$$v := \lfloor (L-1)/160 \rfloor$$
$$w := L - 160 \cdot v .$$

Then perform the following **algorithm**:

1. Let s be a 160-bit string, used as seed for the generation of p.
2. Compute $h := \mathrm{SHA\text{-}1}(s)$
3. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
4. Let z be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to v do
   5.1 Define the 160-bit string s_i to be the binary expansion of the integer $(z+i) \bmod 2^{160}$.
   5.2 Compute $h_i := \mathrm{SHA\text{-}1}(s_i)$.
6. Let h be the string obtained by the concatenation of $h_0, \ldots, h_v$ as follows: $h := h_0 \| h_1 \| \ldots \| h_v$.
7. Let c be the integer whose binary expansion is given by the bit string h.
8. Search for the smallest (probable) prime $p \geq c$ with $p \equiv 3 \bmod 4$. This search can be performed using the function NextPrime implemented in most computer algebra programs.
9. Check that $2^{L-1} - 1 < p < 2^L$, if not then set s := update_seed(s) and goto Step 2.
10. Apply a deterministic primality test, e.g. ECPP, to the probable prime p. If the test does not succeed, then set s := update_seed(s) and goto Step 2.
11. Output p, s.

For the generation of the curves brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, and brainpoolP512r1, and the twists thereof given in Section 10 (for naming conventions see Section 8.1) we use the following values as seeds s in Step 1 for the determination of the respective prime fields.

```
Seed_p_160 := 3243F6A8885A308D313198A2E03707344A409382
Seed_p_192 := 2299F31D0082EFA98EC4E6C89452821E638D0137
Seed_p_224 := 7BE5466CF34E90C6CC0AC29B7C97C50DD3F84D5B
Seed_p_256 := 5B54709179216D5D98979FB1BD1310BA698DFB5A
Seed_p_320 := C2FFD72DBD01ADFB7B8E1AFED6A267E96BA7C904
Seed_p_384 := 5F12C7F9924A19947B3916CF70801F2E2858EFC1
Seed_p_512 := 6636920D871574E69A458FEA3F4933D7E0D95748
```

These seeds are derived as follows: We need seven seeds of 160 bit. Therefore we compute the hexadecimal representation R of $\pi \cdot 2^{1120}$ and divide it into 7 segments of 40 hexadecimal symbols, followed by a "Remainder" i.e.

$$R = \mathrm{Seed\_p\_160} \| \mathrm{Seed\_p\_192} \| \mathrm{Seed\_p\_224} \| \cdots \| \mathrm{Seed\_p\_512} \| \mathrm{Remainder}$$

Using the above algorithm we arrive at the following primes:

```
p_160 = 1332297598440044874827085558802491743757193798159
p_192 = 4781668983906166242955001894344923773259119655253013193367
p_224 =
22721622932454352787552537995910928073340732145944992304435472941311
p_256 =
768849563970453442208097466290016490930379502009430552037356014450315161977
51
p_320 =
176359332223916635416190984244601952088951277271951519277296041528864086880
2149818095501499903527
```

```
p_384 =
21659270770119316173069236842332604979796116387017648600081618503821089934025961822236561982844534088440708417973331
p_512 =
8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947
```

The calculations were done using the ECPP (Elliptic Curve Primality Proving) implementation of Magma (version 2.11-13) on Opteron processors [Mag]. The time consuming ECPP could be used, because the curve generation process was not time critical. The corresponding primality certificates are not included in this paper. Note that there were recent Magma versions with ECPP disabled.

5.2 Generation of pseudo-random curves.

The generation procedure is similar to the procedure given in FIPS PUB 186-2 [FIPS 186-2], Appendix 6.4, and ANSI X9.62 [X9.62], A.3.2.
We use the same update function for seeds as in Section 5.1:

Function update_seed
Input: A 160-bit string *s*.
Let *z* be the integer whose binary expansion is given by the 160-bit string s.
Define the 160-bit string *t* to be the binary expansion of the integer $(z+1) \bmod 2^{160}$.
Output: *t*

Given a prime *p* the following procedure generates a suitable elliptic curve over $GF(p)$.

Let *L* be the bit length of *p*, and define

$$v := \lfloor (L-1)/160 \rfloor$$
$$w := L - 160 \cdot v - 1$$

Then perform the following **algorithm**:

1. Choose an arbitrary 160-bit string *s*.
2. Compute $h := \text{SHA-1}(s)$.
3. Let h_0 be the bit string obtained by taking the *w* rightmost bits of *h*.
4. Let *z* be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to *v* do
   5.1. Define the 160-bit string s_i to be the binary expansion of the integer $(z+i) \bmod 2^{160}$.
   5.2. Compute $h_i := \text{SHA-1}(s_i)$.
6. Let h be the string obtained by the concatenation of $h_0, \ldots, h_v$ as follows:
   $$h := h_0 \| h_1 \| \ldots \| h_v .$$
7. Let A be the integer whose binary expansion is given by the bit string *h*.
8. If $-3 = A \cdot Z^4$ is solvable, then compute and store one solution *Z*, else set *s*:=update_seed(*s*) and goto Step 2. This check can be done using Legendre-symbols and square-root algorithms, as implemented in all computer-algebra systems.
9. Set seed_A := *s*
10. Set *s* := update_seed(*s*)

11. Compute $h := SHA\text{-}1(s)$.
12. Let h_0 be the bit string obtained by taking the *w* rightmost bits of *h*.
13. Let *z* be the integer whose binary expansion is given by the 160-bit string s.
14. For i from 1 to *v* do
    14.1 Define the 160-bit string s_i to be the binary expansion of the integer $(z+i) \bmod 2^{160}$.
    14.2 Compute $h_i := \mathrm{SHA\text{-}1}(s_i)$.
15. Let h be the string obtained by the concatenation of $h_0, \ldots, h_v$ as follows:

$$h := h_0 \| h_1 \| \ldots \| h_v .$$

16. Let B be the integer whose binary expansion is given by the bit string *h*.
17. If B is a square mod *p*, then set *s* := update_seed(*s*) and goto Step 2.
18. Set seed_B:=*s*
19. If $-16(4A^3 + 27B^2) = 0$, then set s := update_seed(*s*) and goto Step 2.
20. Check that the elliptic curve *E* over *GF(p)* given by $y^2 = x^3 + A \cdot x + B$ fulfills all security and functional requirements given in Section 3. If not, then set *s* := update_seed(*s*) and goto Step 2.
21. Set s := update_seed(s).
22. Set seed_BP:=s.
23. Compute $h := SHA\text{-}1(s)$.
24. Let h_0 be the bit string obtained by taking the *w* rightmost bits of *h*.
25. Let *z* be the integer whose binary expansion is given by the 160-bit string s.
26. For i from 1 to *v* do
    26.1 Define the 160-bit string s_i to be the binary expansion of the integer $(z+i) \bmod 2^{160}$.
    26.2 Compute $h_i := SHA\text{-}1(s_i)$.
27. Let h be the string obtained by the concatenation of $h_0, \ldots, h_v$ as follows:

$$h := h_0 \| h_1 \| \ldots \| h_v .$$

28. Let *Mult* be the integer whose binary expansion is given by the bit string *h*.
29. Find a base point P_0 (of order q =#*E(GF(p))* in *E(GF(p)))*. We first choose P as a point of order *q* with smallest *x*-coordinate in *E(GF(p))*. (It is easy to find all points with a given *x*-coordinate as taking square-roots in *GF(p)* is easy.) Then we define $P_0 := Mult \cdot P$. (Of course one could also use *P* as base-point. But the small *x*-coordinate could possibly make the curve vulnerable to side-channel attacks. Therefore we multiply by *Mult* so that the coordinates of P_0 are of equal size.)
30. Prepare a file containing curve data and a file containing data that allow to verify that the curve is „good“. The description of these files is given in Section 8.

For the generation of the curves brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, and brainpoolP512r1, and the twists thereof given in Section 10 (for naming conventions see Section 8.1) we used the following values as initial seeds for the determination of the respective curve parameters *A* and *B*:

```
Seed_ab_160 := 2B7E151628AED2A6ABF7158809CF4F3C762E7160
Seed_ab_192 := F38B4DA56A784D9045190CFEF324E7738926CFBE
Seed_ab_224 := 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7C7B5
Seed_ab_256 := 757F5958490CFD47D7C19BB42158D9554F7B46BC
```

```
Seed_ab_320 := ED55C4D79FD5F24D6613C31C3839A2DDF8A9A276
Seed_ab_384 := BCFBFA1C877C56284DAB79CD4C2B3293D20E9E5E
Seed_ab_512 := AF02AC60ACC93ED874422A52ECB238FEEE5AB6AD
```

These seeds are derived as follows: We need seven seeds of 160 bit, each. Therefore we compute the hexadecimal representation R of $\lfloor e \cdot 2^{1120} \rfloor$ (where e denotes the Euler number 2.718...) and divide it into 7 segments of 40 hexadecimal symbols, followed by a "Remainder", i.e.

$$R = \text{Seed\_ab\_160} \| \text{Seed\_ab\_192} \| \text{Seed\_ab\_224} \| \cdots \| \text{Seed\_ab\_512} \| \text{Remainder}.$$

6. Class group computations

It is well known that the ring of isogenies of *E*, *End(E)*, is (isomorphic to) an order in an imaginary quadratic field $K = Q(\sqrt{-d})$ with square-free $d \in N$. It can be written as $End(E) = Z + c_E O_K$ where Z are the rational integers, O_K is the ring of integers of K and c_E denotes the conductor $[O_K : End(E)]$. The discriminant of $End(E)$ is $c_E^2 \cdot d_K$ where d_K is the discriminant of O_K, i.e.:

$$d_K := \begin{cases} -d & \text{if} \quad -d = 1 \bmod 4 \\ -4d & \text{if} \quad -d = 2 \bmod 4 \quad \text{or} \quad -d = 3 \bmod 4 \end{cases}$$

Let $\# E(GF(p)) = p + 1 + u$, then *d* can be computed as the square-free part of

$$4 \cdot p - u^2 = -c_E^2 \cdot (End(E) : Z[Frob])^2 \cdot d_K,$$

where *Frob* denotes the Frobenius-endomorphism of *E*. For this calculation the factorisation of an *L*-bit number can be necessary.

If $v := \max\{a \mid a^2 \text{ divides } 4 \cdot p - u^2\}$ then $d = (4 \cdot p - u^2)/v^2$.

Since the complexity of the best known algorithms for explicitly determining the class number of *K* is too high in practice one just tries to find elements of the ideal class group of K with a large order, as the class number is not smaller than the order of an element. More information on class group computations is given in Appendix A.

Remark 1. In [JMV] the following remark is made: *„One may consider c(E)*[2] *as a clear way to measure how generic a given curve is. Sometimes in order to convince others that a curve is not specially chosen, one gives the seed of a secure hash based generator for it. However the seed may have been picked with a large number of trials or the hash function may have admitted some compromise. For this reason it may be a good standard practise to reveal c(E), preferably by giving the complete factorization of the discriminant of the characteristic polynomial of Frobenius.“*

Note that our approach yields the data requested in [JMV]. The information is part of the validation data given with every curve. Additionally we describe how our seeds are generated.

[2] In our notation c(E) is denoted by c_E.

7. Security tests

During the curve generation phase all requirements described in Section 3 and Section 4 were tested. We provide data that allow to verify that the curves were pseudo-randomly generated and ease the reproduction of the security proofs.
The main problem in the security proofs is the check of the immunity against the Weil- and Tate-pairing attacks and of the class-number condition. Both checks demand the factorisation of large numbers. These factorisations into prime factors are provided.
Primality Certificates for the respective prime numbers p and q as well as for the alleged prime factors of $q-1$ and $4 \cdot p - u^2$ are not included.

8. Formats

For each curve three sets of data are provided. Two describing the curve, the other one giving validation data. The structure of the data given is as follows:

8.1 Naming conventions

The curves are named as follows: The curve with curve-ID = brainpoolPLrj is the jth curve provided by the ECC-Brainpool over $GF(p)$, where p is an L-bit prime and the coefficient A is selected **r**andomly according to the procedures described in Section 5. The curve brainpoolPLrj is $GF(p)$-isomorphic to the **t**wisted curve brainpoolPLtj with coefficient $A' = -3 \bmod p$.

8.2 Curve Data in descriptive form.

In Section 10 the following data is provided in **hexadecimal** notation for all proposed curves.

Data	**Description**
Curve-ID	Curve-ID as defined in Section 8.1
p	Characteristic of the prime field, i. e. $p = p\_L$ as described in Section 5
A	Coefficient A of the curve equation
B	Coefficient B of the curve equation
$x(P_0)$	x-Coordinate of the base point P_0
$y(P_0)$	y-Coordinate of the base point P_0
q	Order of the cyclic group $\langle P_0 \rangle$ generated by P_0
i	Index of $\langle P_0 \rangle$ in $E(GF(p))$, i.e. $i = \# E(GF(p))/q$. (i = 1 follows from our requirements for all Brainpool-Curves.)
#Twisted curve	
Curve-ID	Curve-ID of the twisted curve with $A' = -3 \bmod p$
Z	Twist with Z^2 leads to the $GF(p)$-isomorphic curve $y^2 = x^3 + A'x + B'$ with base-point P_0'
A'	$A' = Z^4 A = -3 \bmod p$
B'	$B' = Z^6 B$
$x(P_0')$	$x(P_0') = x(P_0) \cdot Z^2$
$y(P_0')$	$y(P_0') = y(P_0) \cdot Z^3$

8.3 Curve Data in ASN.1 syntax.

Appendix B describes the ASN.1 syntax that can be used to describe elliptic curves. The ASN.1 coding according to X9.62 [X9.62] of all proposed curves is provided in this appendix. In addition Appendix B gives OIDs for all curves.

8.4 Validation Data.

The validation data is structured as described below. All values are given in decimal or in hexadecimal notation as indicated.
The validation data for the curves brainpoolPLrj and brainpoolPLtj are given with brainpoolPL*j-Eval. This is possible as both curves are isomorphic.

Data	**Description**
Seed	Seed_ab_L used in the algorithm described in Section 5
Seed_A	Seed_A defined in the algorithm described in Section 5
Seed_B	Seed_B defined in the algorithm described in Section 5
#Comment:	$\#E(GF(p)) - p - 1 = u$ and $4 \cdot p = u^2 + d \cdot v^2$
u	$\#E(GF(p)) - p - 1$
v	Value v as defined in Section 6. Derived from the factorisation of $4 \cdot p - u^2$.
d	Value d as defined in Section 6. Derived from the factorisation of $4 \cdot p - u^2$.
#Factorisation of *d*.	
t	Number of different prime factors of d
factor_1	
$\vdots$	Prime factors of d
factor_t	
$-d \bmod 4$	
#Weil-Tate-Bound	
#Factorisation of $q-1$	This factorisation is used to compute the order of $p \bmod q$ using the fact that the order of every element of $GF(q)$ divides $q-1$
r	number of different prime factors of $q-1$
factor_1	
exponent_1	
$\vdots$	
factor_r	
exponent_r	
$(q-1)$ /(order of $p \bmod q$)	Bound has to be < 100 by Requirement 1 of Section 4.

#Class number bound	
#quadratic form	quadratic form (qf_a, qf_b, qf_c) with discriminant $(qf\_b)^2 - 4 \cdot qf\_a \cdot qf\_c = -d$ if $-d \equiv 1 \bmod 4$ and $(qf\_b)^2 - 4 \cdot qf\_a \cdot qf\_c = -4 \cdot d$ if $-d \equiv 2,3 \bmod 4$ Such a form can easily be found. For example one can fix qf_b and find small factors of $(qf\_b)^2 + (4) \cdot d$
qf_a	
qf_b	
qf_c	
MinClass	Lower bound for the order of the quadratic form (qf_a, qf_b, qf_c).
#Basepoint-Construction	
x(P)	*x*-coordinate of P (Hex)
y(P)	**y**-coordinate of P (Hex)
Mult	As defined in Section 5. (Hex)
Seed_BP	Seed for generating Mult. (Hex)

9. References:

[BJ] E. Brier, M. Joyce, Fast multiplication on Elliptic Curves through Isogenies. In: M. Fossorier, T. Hoholdt, and A. Poli, eds., *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Lecture Notes in Computer Science **2643**, Springer 2003.

[BP] http://www.ecc-brainpool.org

[BG] D. R. L. Brown, R. P. Gallant, The static Diffie-Hellman Problem, http://www.iacr.org/2004/306, version dated June 23, 2005

[BRS] J. Bohli, S. Röhrich, R. Steinwandt. Key substitution attacks revisited: taking into account malicious signers, preprint, 2004.

[BSI] http://www.bsi.bund.de

[BSS] I. Blake, G. Seroussi, N. Smart, Elliptic Curves in Cryptography, *Cambridge University press 1999*, ISBN 0-521-65374-6.

[CF] H. Cohen, G. Frey et. al., Handbook of Elliptic and Hyperelliptic Curve Cryptography, *Chapman & Hall/CRC 2006*, ISBN 1-58488-518-1

[Coh] H. Cohen, A course in computational algebraic number theory, *Springer 1993*, ISBN 3-540-55640-0.

[Cox] D. A. Cox, Primes of the form $x^2 + n \cdot y^2$, Wiley, 1989, ISBN 0-471-50654-0

[DUM] http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c

[FIPS 186-2] NIST, FIPS Publication 186-2, Digital Signature Standard (DSS), 2000 and change notice 1, 2001. http://www.csrc.nist.gov

[G] L. Goubin. A refined power-analysis-attack on Elliptic Curve Cryptosystems. In: Public-Key-Cryptography – PKC2003, Lecture Notes in Computer Science, **2567,** Springer 2003.

[HMCD] Y. Hitchcock, P. Montague, G. Carter, E. Dawson, The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves. International Journal of Information Security **3**, 86-98, 2004.

[HMV] D. Hankerson, A. Menezes, S. Vanstone. Guide to Elliptic Curve Cryptography. *Springer 2004*, ISBN 0-387-95273-X.

[HR] Ming-Deh Huang and Wayne Raskind. Global methods for discrete logarithm problems. *Slides of a talk presented at ECC 2004*, accessible via http://www.cacr.math.uwaterloo.ca/conferences/2004/ecc2004/slides.html

[ICAO] http://www.icao.int/mrtd

[ISO] ISO/IEC FDIS 18032

[JMV] D. Jao, S. D. Miller, R. Venkatesan, Ramanujan graphs and the random reducibility of discrete log on isogenous elliptic curves, accessible via the IACR preprint server http://www.iacr.org/2004/312

[KS] F. Kuhn, R. Struik, Random walks revisited: Extensions of Pollard's Rho algorithm for computing multiple discrete logarithms. In: Selected Areas in Cryptography – SAC 2001. Lecture Notes in Computer Science, **2259**. *Springer 2001*, 212-229.

[Mag] http://magma.maths.usyd.edu.au/magma

[NIST] Draft NIST Special Publication 800-57, Recommendations for Key-Management http://csrc.nist.gov/Crypto-Toolkit/kms/guideline-1-Jan03.pdf

[SA] T. Satoh, K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Comm. Math. Univ. Sancti Pauli*, **47**, 81-92, 1998.

[SEC 1] SEC 1. Standards for Efficient Cryptography Group: Elliptic Curve Cryptography. Version 1.0, 2000. http://www.secg.org/download/aid-385/sec1_final.pdf

[SEC 2] SEC 2. Standards for Efficient Cryptography Group: Recommended Elliptic Curve Domain Parameters. Version 1.0, 2000. http://www.secg.org/download/aid-386/sec2_final.pdf

[Sem] I. A. Semaev. Evaluation of discrete logarithms on some elliptic curves. *Math. Comp.*, **67**, 353-356, 1998.

[Sil] J. H. Silverman. The arithmetic of elliptic curves. Springer 1986, ISBN 3-540-96203-4.

[Sma] N. P. Smart. The discrete logarithm problem on elliptic curves of trace one. *J. Cryptology* **12**, 193-196, 1999.

[Spa] A. M. Spallek. Konstruktion einer elliptischen Kurve über einem endlichen Körper zu gegebener Punktegruppe. *Diplomarbeit*, Essen, 1992.

[TTT] http://www.teletrust.de

[X9.62] ANSI X9.62, 1998.

10. Curve Data

10.1 160 bit curves

```
Curve-ID: brainpoolP160r1
p: E95E4A5F737059DC60DFC7AD95B3D8139515620F
A: 340E7BE2A280EB74E2BE61BADA745D97E8F7C300
B: 1E589A8595423412134FAA2DBDEC95C8D8675E58
x(P_0): BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3
y(P_0): 1667CB477A1A8EC338F94741669C976316DA6321
q: E95E4A5F737059DC60DF5991D45029409E60FC09
i: 1


#Twisted curve
Curve-ID: brainpoolP160t1
Z: 24DBFF5DEC9B986BBFE5295A29BFBAE45E0F5D0B
A': E95E4A5F737059DC60DFC7AD95B3D8139515620C
B': 7A556B6DAE535B7B51ED2C4D7DAA7A0B5C55F380
x(P_0'): B199B13B9B34EFC1397E64BAEB05ACC265FF2378
y(P_0'): ADD6718B7C7C1961F0991B842443772152C9E0AD
```

10.2 192 bit curves

```
Curve-ID: brainpoolP192r1
p: C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297
A: 6A91174076B1E0E19C39C031FE8685C1CAE040E5C69A28EF
B: 469A28EF7C28CCA3DC721D044F4496BCCA7EF4146FBF25C9
x(P_0): C0A0647EAAB6A48753B033C56CB0F0900A2F5C4853375FD6
y(P_0): 14B690866ABD5BB88B5F4828C1490002E6773FA2FA299B8F
q: C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1
i: 1

#Twisted curve
Curve-ID: brainpoolP192t1
Z: 1B6F5CC8DB4DC7AF19458A9CB80DC2295E5EB9C3732104CB
A': C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86294
B': 13D56FFAEC78681E68F9DEB43B35BEC2FB68542E27897B79
x(P_0'): 3AE9E58C82F63C30282E1FE7BBF43FA72C446AF6F4618129
y(P_0'): 97E2C5667C2223A902AB5CA449D0084B7E5B3DE7CCC01C9
```

10.3 224 bit curves

```
Curve-ID: brainpoolP224r1
p: D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF
A: 68A5E62CA9CE6C1C299803A6C1530B514E182AD8B0042A59CAD29F43
B: 2580F63CCFE44138870713B1A92369E33E2135D266DBB372386C400B
x(P_0): D9029AD2C7E5CF4340823B2A87DC68C9E4CE3174C1E6EFDEE12C07D
y(P_0): 58AA56F772C0726F24C6B89E4ECDAC24354B9E99CAA3F6D3761402CD
q: D7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F
i: 1


#Twisted curve
Curve-ID: brainpoolP224t1
Z: 2DF271E14427A346910CF7A2E6CFA7B3F484E5C2CCE1C8B730E28B3F
A': D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FC
B': 4B337D934104CD7BEF271BF60CED1ED20DA14C08B3BB64F18A60888D
x(P_0'): 6AB1E344CE25FF3896424E7FFE14762ECB49F8928AC0C76029B4D580
y(P_0'): 374E9F5143E568CD23F3F4D7C0D4B1E41C8CC0D1C6ABD5F1A46DB4C
```

10.4 256 bit curves

```
Curve-ID: brainpoolP256r1
p: A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
A: 7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
B: 26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6
x(P_0): 8BD2AEB9CB7E57CB2C4B482FFC81B7AFB9DE27E1E3BD23C23A4453BD9ACE3262
y(P_0): 547EF835C3DAC4FD97F8461A14611DC9C27745132DED8E545C1D54C72F046997
q: A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7
i: 1


#Twisted curve
Curve-ID: brainpoolP256t1
Z: 3E2D4BD9597B58639AE7AA669CAB9837CF5CF20A2C852D10F655668DFC150EF0
A': A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5374
B': 662C61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04
x(P_0'): A3E8EB3CC1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F4
y(P_0'): 2D996C823439C56D7F7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE
```

10.5 320 bit curves

```
Curve-ID: brainpoolP320r1
p:
D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B
32E27
A:
3EE30B568FBAB0F883CCEBD46D3F3BB8A2A73513F5EB79DA66190EB085FFA9F492F375A97D8
60EB4
B:
520883949DFDBC42D3AD198640688A6FE13F41349554B49ACC31DCCD884539816F5EB4AC8FB
1F1A6
x(P_0):
43BD7E9AFB53D8B85289BCC48EE5BFE6F20137D10A087EB6E7871E2A10A599C710AF8D0D39E
20611
y(P_0):
14FDD05545EC1CC8AB4093247F77275E0743FFED117182EAA9C77877AAAC6AC7D35245D1692
E8EE1
q:
D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C
59311
i: 1


#Twisted curve
Curve-ID: brainpoolP320t1
Z:
15F75CAF668077F7E85B42EB01F0A81FF56ECD6191D55CB82B7D861458A18FEFC3E5AB7496F
3C7B1
A':
D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B
32E24
B':
A7F561E038EB1ED560B3D147DB782013064C19F27ED27C6780AAF77FB8A547CEB5B4FEF4223
40353
x(P_0'):
925BE9FB01AFC6FB4D3E7D4990010F813408AB106C4F09CB7EE07868CC136FFF3357F624A21
BED52
y(P_0'):
63BA3A7A27483EBF6671DBEF7ABB30EBEE084E58A0B077AD42A5A0989D1EE71B1B9BC0455FB
0D2C3
```

10.6 384 bit curves

```
Curve-ID: brainpoolP384r1
p:
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901
D1A71874700133107EC53
A:
7BC382C63D8C150C3C72080ACE05AFA0C2BEA28E4FB22787139165EFBA91F90F8AA5814A503
AD4EB04A8C7DD22CE2826
B:
4A8C7DD22CE28268B39B55416F0447C2FB77DE107DCD2A62E880EA53EEB62D57CB4390295DB
C9943AB78696FA504C11
x(P_0):
1D1C64F068CF45FFA2A63A81B7C13F6B8847A3E77EF14FE3DB7FCAFE0CBD10E8E826E03436D
646AAEF87B2E247D4AF1E
y(P_0):
8ABE1D7520F9C2A45CB1EB8E95CFD55262B70B29FEEC5864E19C054FF99129280E464621779
1811142820341263C5315
q:
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7
FC3103B883202E9046565
i: 1


#Twisted curve
Curve-ID: brainpoolP384t1
Z:
41DFE8DD399331F7166A66076734A89CD0D2BCDB7D068E44E1F378F41ECBAE97D2D63DBC87B
CCDDCCC5DA39E8589291C
A':
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901
D1A71874700133107EC50
B':
7F519EADA7BDA81BD826DBA647910F8C4B9346ED8CCDC64E4B1ABD11756DCE1D2074AA263B8
8805CED70355A33B471EE
x(P_0'):
18DE98B02DB9A306F2AFCD7235F72A819B80AB12EBD653172476FECD462AABFFC4FF191B946
A5F54D8D0AA2F418808CC
y(P_0'):
25AB056962D30651A114AFD2755AD336747F93475B7A1FCA3B88F2B6A208CCFE469408584DC
2B2912675BF5B9E582928
```

10.7 512 bit curves

```
Curve-ID: brainpoolP512r1
p:
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC
66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3
A:
7830A3318B603B89E2327145AC234CC594CBDD8D3DF91610A83441CAEA9863BC2DED5D5AA82
53AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CA
B:
3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C
1AC4D77FC94CADC083E67984050B75EBAE5DD2809BD638016F723
x(P_0):
81AEE4BDD82ED9645A21322E9C4C6A9385ED9F70B5D916C1B43B62EEF4D0098EFF3B1F78E2D
0D48D50D1687B93B97D5F7C6D5047406A5E688B352209BCB9F822
y(P_0):
7DDE385D566332ECC0EABFA9CF7822FDF209F70024A57B1AA000C55B881F8111B2DCDE494A5
F485E5BCA4BD88A2763AED1CA2B2FA8F0540678CD1E0F3AD80892
q:
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA
92619418661197FAC10471DB1D381085DDADDB58796829CA90069
i: 1


#Twisted curve
Curve-ID: brainpoolP512t1
Z:
12EE58E6764838B69782136F0F2D3BA06E27695716054092E60A80BEDB212B64E585D90BCE1
3761F85C3F1D2A64E3BE8FEA2220F01EBA5EEB0F35DBD29D922AB
A':
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC
66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F0
B':
7CBBBCF9441CFAB76E1890E46884EAE321F70C0BCB4981527897504BEC3E36A62BCDFA23049
76540F6450085F2DAE145C22553B465763689180EA2571867423E
x(P_0'):
640ECE5C12788717B9C1BA06CBC2A6FEBA85842458C56DDE9DB1758D39C0313D82BA51735CD
B3EA499AA77A7D6943A64F7A3F25FE26F06B51BAA2696FA9035DA
y(P_0'):
5B534BD595F5AF0FA2C892376C84ACE1BB4E3019B71634C01131159CAE03CEE9D9932184BEE
F216BD71DF2DADF86A627306ECFF96DBB8BACE198B61E00F8B332
```

11. Validation Data

11.1 brainpoolP160*1-Eval

```
seed: 2B7E151628AED2A6ABF7158809CF4F3C762E7160
seed_A: 2B7E151628AED2A6ABF7158809CF4F3C762E727A
seed_B: 2B7E151628AED2A6ABF7158809CF4F3C762E727D

u: -519972310379544251229703
v: 33
d: 464538033994374508452344387283800832672277844 3
```

```
#Factorisation of d:
Number of different prime factors of d: 6
Factor_1: 17
Factor_2: 29
Factor_3: 89
Factor_4: 22067
Factor_5: 577011261754261
Factor_6: 8314894957527277176257
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 9
Factor_1: 2
Exponent_1: 3

Factor_2: 3
Exponent_2: 1

Factor_3: 83
Exponent_3: 1

Factor_4: 1933
Exponent_4: 1

Factor_5: 216841
Exponent_5: 1

Factor_6: 2745161
Exponent_6: 1

Factor_7: 3244753
Exponent_7: 1

Factor_8: 72663031601
Exponent_8: 1

Factor_9: 2465333512157
Exponent_9: 1

(q - 1) / (order of p mod q): 3

#Class number bound
#quadratic form:
qf_a: 3
```

```
qf_b: 1
qf_c: 387115028328645423710286989403167360560231537
MinClass: 10000000
#Basepoint-Construction

x(P): 2
y(P): 3FBC64E9988D7CC563721D4116184EDB2656E688
Mult: 2187040EA6E6EC5D867AB235A349A55BAA5E9C32
seed_BP: 2B7E151628AED2A6ABF7158809CF4F3C762E727E
```

*11.2 brainpoolP192*1-Eval*

```
seed: F38B4DA56A784D9045190CFEF324E7738926CFBE
seed_A: F38B4DA56A784D9045190CFEF324E7738926D4A4
seed_B: F38B4DA56A784D9045190CFEF324E7738926D4A5

u: -75885465139255996133178324439
v: 1
d: 13368072116223427911218896962387160374571840032632508108747

#Factorisation of d:
Number of different prime factors of d: 7
Factor_1: 3
Factor_2: 17
Factor_3: 19
Factor_4: 47
Factor_5: 103
Factor_6: 11689
Factor_7: 243799360126346034321229132107730224949211782787
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 5
Factor_1: 2
Exponent_1: 6

Factor_2: 3
Exponent_2: 1

Factor_3: 17
Exponent_3: 1

Factor_4: 13609004849343556497893651
Exponent_4: 1

Factor_5: 107647262337333555283688982427
Exponent_5: 1

(q - 1) / (order of p mod q): 8

#Class number bound
#quadratic form:
qf_a: 7
qf_b: 5
qf_c: 477431147007979568257817748656684299091851429736875289599
MinClass: 10000000
#Basepoint-Construction

x(P): 2
y(P): 1E2412C2512EFCF3747B221CCEE8C15304BF387BA1810033
Mult: 6FBF25C9A6392E5353EB6D02255D716E4043DA7816C55490
seed_BP: F38B4DA56A784D9045190CFEF324E7738926D4A6
```

11.3 brainpoolP224*1-Eval

```
seed: 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7C7B5
seed_A: 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7CCF5
seed_B: 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7CCF8

u: -4460773185803614235113038911974753
v: 1
d: 70987994314632885262411297299970290552009893845341623260563782354235

#Factorisation of d:
Number of different prime factors of d: 5
Factor_1: 3
Factor_2: 5
Factor_3: 443
Factor_4: 262401365399
Factor_5: 40712130504760273201793920999767203487306743499252457
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 4
Factor_1: 2
Exponent_1: 1

Factor_2: 3
Exponent_2: 1

Factor_3: 173
Exponent_3: 1

Factor_4: 21889810146873172242343485545193568027521720946368744885738498041
Exponent_4: 1

(q - 1) / (order of p mod q): 6

#Class number bound
#quadratic form:
qf_a: 1531
qf_b: 7
qf_c: 11591769156537048540563569121484371416069545043328155333207671841
MinClass: 10000000
#Basepoint-Construction

x(P): 1
y(P): 4EBC9078AD8AD07562CD41B374827192AA88CE3C718A014405EED475
Mult: 66DBB372386C400BE646C1B80C4A40580359B836DFD41B5485953527
seed_BP: 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7CCF9
```

*11.4 brainpoolP256*1-Eval*

seed: 757F5958490CFD47D7C19BB42158D9554F7B46BC
seed_A: 757F5958490CFD47D7C19BB42158D9554F7B4E51
seed_B: 757F5958490CFD47D7C19BB42158D9554F7B4E52

u: -300418416528525664980082381967979838673
v: 5
d:
869154402394698537706294265970853537179110317908890724162791987506875833560
3

#Factorisation of d:
Number of different prime factors of d: 4
Factor_1: 1867
Factor_2: 11616307
Factor_3: 66891682553
Factor_4: 5991180651865208777371442029237375648904065304322611579
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 3
Factor_1: 2
Exponent_1: 1

Factor_2: 3
Exponent_2: 2

Factor_3:
427138646650251912337831925716675828292986287691191830770863994794797424217
1
Exponent_3: 1

(q - 1) / (order of p mod q): 2

#Class number bound
#quadratic form:
qf_a: 11
qf_b: 1
qf_c:
197535091453340576751430514993375803904343254070202437309725451706108143991
MinClass: 10000000
#Basepoint-Construction

x(P): 1
y(P): 9E0E9E8D98FB89DA2A32B2C7618B26BB99B920F02A5E831A142E6C8673110CD
Mult: 5CF7E1CE6BCCDC18FF8C07B6E9B89F067C39996241690B7C6FF4A4CF27CE38F7
seed_BP: 757F5958490CFD47D7C19BB42158D9554F7B4E53

*11.5 brainpoolP320*1-Eval*

```
seed: ED55C4D79FD5F24D6613C31C3839A2DDF8A9A276
seed_A: ED55C4D79FD5F24D6613C31C3839A2DDF8A9AB24
seed_B: ED55C4D79FD5F24D6613C31C3839A2DDF8A9AB29

u: -1829129012274291271856083956306349739816241699607
v: 1
d:
370866034541314102639171916159228059107613859651279075961026522257163607968
7307055058681975659659

#Factorisation of d:
Number of different prime factors of d: 3
Factor_1: 877
Factor_2: 3111160968660033089
Factor_3:
135923636644265746730732953285333242292531722023077912660283440140765492503
3
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 8
Factor_1: 2
Exponent_1: 4

Factor_2: 5
Exponent_2: 1

Factor_3: 7
Exponent_3: 1

Factor_4: 13
Exponent_4: 1

Factor_5: 89
Exponent_5: 1

Factor_6: 317
Exponent_6: 1

Factor_7: 28661
Exponent_7: 1

Factor_8:
299589410500344960818634453344053290288849906371853185689121682914766027672
438458623
Exponent_8: 1

(q - 1) / (order of p mod q): 1

#Class number bound
#quadratic form:
qf_a: 31
qf_b: 3
qf_c:
299085511726866211805783803354216176699688596492966996742763324400938393523
1699237950549980370 7
MinClass: 10000000
```

```
#Basepoint-Construction

x(P): 1
y(P):
29110253D52CF3C5FC3382FCA93D18ADF7B97999028767B9722381DB68FE3A41793B7D9952C
6177F
Mult:
1554B49ACC31DCCD884539816F5EB4AC8FB1F1A6D41318159E53765CD93C0457DF9FAF9A000
C6538
seed_BP: ED55C4D79FD5F24D6613C31C3839A2DDF8A9AB2A
```

11.6 brainpoolP384*1-Eval

```
seed: BCFBFA1C877C56284DAB79CD4C2B3293D20E9E5E
seed_A: BCFBFA1C877C56284DAB79CD4C2B3293D20EB473
seed_B: BCFBFA1C877C56284DAB79CD4C2B3293D20EB475

u: -5973228999478432667446284273866865475630836211106699249391
v: 11
d:
4211373421509681564010736574639026085179555808599119887457074293277751733352
4566815471557566819175754457870318844483

#Factorisation of d:
Number of different prime factors of d: 4
Factor_1: 6857
Factor_2: 379795560371
Factor_3: 3757338063608181501812427899557428249340 1057
Factor_4: 4303872990401223419424669070518309027547364146662699977
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 9
Factor_1: 2
Exponent_1: 2

Factor_2: 5
Exponent_2: 1

Factor_3: 7
Exponent_3: 2

Factor_4: 6997
Exponent_4: 1

Factor_5: 124822877
Exponent_5: 1

Factor_6: 657021949
Exponent_6: 1

Factor_7: 53014379452169
Exponent_7: 1

Factor_8: 8067780739605749
Exponent_8: 1

Factor_9: 9005006809066404255164693630348690272461585532331405760419 3773
Exponent_9: 1

(q - 1) / (order of p mod q): 4

#Class number bound
#quadratic form:
qf_a: 7
qf_b: 3
qf_c:
1504061936253457701432405919513937887564127887853999598060979618849113333340
163100552555627386399134087781082815 89
```

```
MinClass: 10000000
#Basepoint-Construction

x(P): 1
y(P):
44E54365091651EEBE3AA1E13A14EC2C0DD1B1AD3778F69D586D078D7554C116A71E422ADD5
1CEA477CE154CE873940E
Mult:
7DCD2A62E880EA53EEB62D57CB4390295DBC9943AB78696FA504C115037CD644E494DCC245B
3B8813113DD705F4C2C3
seed_BP: BCFBFA1C877C56284DAB79CD4C2B3293D20EB476
```

*11.7 brainpoolP512*1-Eval*

```
seed: AF02AC60ACC93ED874422A52ECB238FEEE5AB6AD
seed_A: AF02AC60ACC93ED874422A52ECB238FEEE5ABDFC
seed_B: AF02AC60ACC93ED874422A52ECB238FEEE5ABDFD

u: -
133911538952573548431982907995132016398065354869678696041884562716142492272
779
v: 1
d:
178635485659543074419414068896560771556679588088352071164904017023247289957
846082383474428018735919180202445618412767323363477951441605998809524134868
84947

#Factorisation of d:
Number of different prime factors of d: 3
Factor_1: 557
Factor_2: 407446053312479106780211110971118243100813139950432927
Factor_3:
787122676748868966456205356775558572976331852829894157046175701241383459173
65678302464602380771873
-d mod 4: 1

#Weil-Tate-Bound
#Factorisation of q-1

Number of different prime factors of q-1: 12
Factor_1: 2
Exponent_1: 3

Factor_2: 3
Exponent_2: 1

Factor_3: 41
Exponent_3: 1

Factor_4: 1559
Exponent_4: 1

Factor_5: 4391
Exponent_5: 1

Factor_6: 14557
Exponent_6: 1

Factor_7: 33278821
Exponent_7: 1

Factor_8: 56951731
Exponent_8: 1

Factor_9: 82441673
Exponent_9: 1

Factor_10: 447825179
Exponent_10: 1

Factor_11: 55886090402833
Exponent_11: 1
```

```
Factor_12:
233379254077089173432698734933196076388793414859142247462280385464743821834
28298476287118961279
Exponent_12: 1

(q - 1) / (order of p mod q): 6

#Class number bound
#quadratic form:
qf_a: 13
qf_b: 1
qf_c:
343529780114505912345027055570309176070537669400677059932507725044706326842
011696891296976959107536885004703112332244852622072983541549997710623336286
249
MinClass: 10000000
#Basepoint-Construction

x(P): 3
y(P):
45DD3AD1B6A380EFF32BCCD947957F3ACD60D5A6DF18ED9A4D676C1924123576C959AE8473D
E224CA262D456E8D51F6DA36EAAE8E3DFC0E914AFDB1BC552796
Mult:
A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CADC083E67984050B75EBAE5DD2809
BD638016F723707F59380B759E9BCE57ACFDA9CB96AC38A433A6
seed_BP: AF02AC60ACC93ED874422A52ECB238FEEE5ABDFE
```

Appendix A. Class group computations.

We use the well known bijection between the class groups of binary quadratic forms with discriminant $d_K < 0$ [3] and the ideal-class group of the order with discriminant d_K (cf. [Coh], [Cox]). For this purpose we represent binary quadratic forms $ax^2 + bxy + gy^2$ as triples (a, b, g).

A triple (a, b, g) of integers is called a ***positive definite primitive reduced binary quadratic form[4] of discriminant*** d_K if:

- $\gcd(a, b, g) = 1$ ("primitive")
- $a > 0$ and $|b| \leq a \leq g$ and if $a = g$ or $|b| = a$ then also $b \geq 0$ ("reduced")
- $b^2 - 4ag = d_K$ ("positive definite")

The elements of the ideal class group of the number field K:=Q($\sqrt{-d}$) with discriminant d_K (see Section 6) correspond bijectively to the primitive reduced quadratic forms of discriminant d_K. The group operation in this set can be calculated as follows:

Given two primitive reduced quadratic forms (a_1, b_1, g_1) and (a_2, b_2, g_2) the so called composition (a', b', g') of (a_1, b_1, g_1) and (a_2, b_2, g_2) can be determined by Algorithm 5.4.7 of Cohen´s book [Coh]. (a', b', g') is primitive and has discriminant d_K but it is not necessarily reduced. The reduction Algorithm 5.4.2 of [Coh], applied to (a', b', g'), outputs a primitive reduced quadratic form (a, b, g) with discriminant d_K. This triple "is" the product of the two elements represented by (a_1, b_1, g_1) and (a_2, b_2, g_2). We denote the multiplication of quadratic forms by $\bullet$, i. e. $(a, b, g) = (a_1, b_1, g_1) \bullet (a_2, b_2, g_2)$.

The neutral element **I** is represented by the triple (1, 0, $-d_K/4$) if $d_K \equiv 0 \bmod 4$ and it is represented by the triple (1, 1, $(1 - d_K)/4$) if $d_K \equiv 1 \bmod 4$.

The following algorithm determines whether the order of an element of the ideal class group of the number field K:=Q($\sqrt{-d}$) has an order of at least a value MinClass[5]:

Input: A primitive reduced quadratic form (a, b, g) of discriminant d_K.

Output: „true“ if the order of the corresponding element of the ideal class group is at least MinClass; „false“ otherwise.

1. Set t := **I**.
2. For i from 1 to MinClass-1 do
   Set $t := t \bullet (a, b, g)$.
   If t = **I** then output "false" and stop.
3. Output „true“.

Note. Most of the common computer algebra packages contain implementations for the described manipulations in the class group of K or in the set of primitive reduced quadratic forms respectively.

---

[3] The theory also holds for imaginary-quadratic orders, but we do not need this fact here and work directly with the maximal order.

[4] In what follows we abbreviate this as „quadratic form“.

[5] All elliptic curves proposed in this paper have MinClass=10000000.

Appendix B. ASN.1 Syntax

B.1 Introduction

This Section provides the syntax for the elliptic curve cryptography domain parameters of the present document according to Abstract Syntax Notation One (ASN.1). The ASN.1 syntax for representing elliptic curve domain parameters is chosen according to ANSI X9.62 [X9.62], 6.3. It is assumed that the reader is familiar with this document.

The object identifier that indicates that elliptic curves defined over prime fields are used is taken from this standard and has the following value:

```
prime-field OBJECT IDENTIFIER ::= {ansi-x9-62 fieldType(1) 1}
```

As an example for one set of domain parameters contained in ANSI X9.62 the encoding for a specific set of domain parameter is given below. The values for this curve are

```
Curve-ID: ANSIp192r1
p:          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF
A:          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC
B:          64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
x(P_0):     188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012
y(P_0):     7192B95FFC8DA78631011ED6B24CDD573F977A11E794811
q:          FFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
i:          1
```

The dump of the encoding for this set of domain parameters (the encoding rules used here and in the following are the Distinguished Encoding Rules (DER)) is:

```
   0 30  176: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   36:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   25:     INTEGER
            :       00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
            :       FE FF FF FF FF FF FF FF FF
            :     }
  44 30   52:   SEQUENCE {
  46 04   24:     OCTET STRING
            :       FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FE
            :       FF FF FF FF FF FF FF FC
  72 04   24:     OCTET STRING
            :       64 21 05 19 E5 9C 80 E7 0F A7 E9 AB 72 24 30 49
            :       FE B8 DE EC C1 46 B9 B1
            :     }
  98 04   49:   OCTET STRING
            :     04 18 8D A8 0E B0 30 90 F6 7C BF 20 EB 43 A1 88
            :     00 F4 FF 0A FD 82 FF 10 12 07 19 2B 95 FF C8 DA
            :     78 63 10 11 ED 6B 24 CD D5 73 F9 77 A1 1E 79 48
            :     11
 149 02   25:   INTEGER
            :     00 FF FF FF FF FF FF FF FF FF FF FF FF 99 DE F8
            :     36 14 6B C9 B1 B4 D2 28 31
 176 02    1:   INTEGER 1
            :   }
```

Note. Although the ASN.1 syntax described in ANSI X9.62 comprises an optional value **`seed`** to derive the coefficients of a randomly generated elliptic curve, this value is not used here. The reason is, that the algorithm for generating the coefficients of the elliptic curves described in this paper is different from the one described in ANSI X9.62.

B.2 OIDs for domain parameters defined in this paper

The object identifier representing the root of the tree containing all object identifiers defined in this paper is given by

```
ecStdCurvesAndGeneration OBJECT IDENTIFIER::= {iso(1) identifified-
organization(3) teletrust(36) algorithm(3) signature-algorithm(3)
ecSign(2) 8}
```

The object identifier **`ellipticCurve`** represents the tree containing the object identifiers for each set of domain parameters specified in this paper. It has the following value:

```
ellipticCurve OBJECT IDENTIFIER ::= {ecStdCurvesAndGeneration 1}
```

The tree for the domain parameters defined in this version of the paper is

```
versionOne OBJECT IDENTIFIER ::= {ellipticCurve 1}
```

The following object identifiers represent the domain parameters defined in this paper:

```
brainpoolP160r1 OBJECT IDENTIFIER ::= {versionOne 1}
brainpoolP160t1 OBJECT IDENTIFIER ::= {versionOne 2}
brainpoolP192r1 OBJECT IDENTIFIER ::= {versionOne 3}
brainpoolP192t1 OBJECT IDENTIFIER ::= {versionOne 4}
brainpoolP224r1 OBJECT IDENTIFIER ::= {versionOne 5}
brainpoolP224t1 OBJECT IDENTIFIER ::= {versionOne 6}
brainpoolP256r1 OBJECT IDENTIFIER ::= {versionOne 7}
brainpoolP256t1 OBJECT IDENTIFIER ::= {versionOne 8}
brainpoolP320r1 OBJECT IDENTIFIER ::= {versionOne 9}
brainpoolP320t1 OBJECT IDENTIFIER ::= {versionOne 10}
brainpoolP384r1 OBJECT IDENTIFIER ::= {versionOne 11}
brainpoolP384t1 OBJECT IDENTIFIER ::= {versionOne 12}
brainpoolP512r1 OBJECT IDENTIFIER ::= {versionOne 13}
brainpoolP512t1 OBJECT IDENTIFIER ::= {versionOne 14}
```

For each set of these domain parameters, the dump of the encoded parameters is given in Subsection B.3.

Note. Future versions of this paper may define additional sets of domain parameters.

B.3 Dump for domain parameters defined in this paper

```
Curve-ID:  brainpoolP160r1

   0 30  152: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   32:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
```

```
 17 02   21:      INTEGER
           :        00 E9 5E 4A 5F 73 70 59 DC 60 DF C7 AD 95 B3 D8
           :        13 95 15 62 0F
           :      }
 40 30   44:    SEQUENCE {
 42 04   20:      OCTET STRING
           :        34 0E 7B E2 A2 80 EB 74 E2 BE 61 BA DA 74 5D 97
           :        E8 F7 C3 00
 64 04   20:      OCTET STRING
           :        1E 58 9A 85 95 42 34 12 13 4F AA 2D BD EC 95 C8
           :        D8 67 5E 58
           :      }
 86 04   41:    OCTET STRING
           :      04 BE D5 AF 16 EA 3F 6A 4F 62 93 8C 46 31 EB 5A
           :      F7 BD BC DB C3 16 67 CB 47 7A 1A 8E C3 38 F9 47
           :      41 66 9C 97 63 16 DA 63 21
129 02   21:    INTEGER
           :      00 E9 5E 4A 5F 73 70 59 DC 60 DF 59 91 D4 50 29
           :      40 9E 60 FC 09
152 02    1:    INTEGER 1
           :    }
```

Curve-ID: brainpoolP160t1

```
  0 30  152: SEQUENCE {
  3 02    1:   INTEGER 1
  6 30   32:   SEQUENCE {
  8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
 17 02   21:     INTEGER
           :       00 E9 5E 4A 5F 73 70 59 DC 60 DF C7 AD 95 B3 D8
           :       13 95 15 62 0F
           :     }
 40 30   44:   SEQUENCE {
 42 04   20:     OCTET STRING
           :       E9 5E 4A 5F 73 70 59 DC 60 DF C7 AD 95 B3 D8 13
           :       95 15 62 0C
 64 04   20:     OCTET STRING
           :       7A 55 6B 6D AE 53 5B 7B 51 ED 2C 4D 7D AA 7A 0B
           :       5C 55 F3 80
           :     }
 86 04   41:   OCTET STRING
           :     04 B1 99 B1 3B 9B 34 EF C1 39 7E 64 BA EB 05 AC
           :     C2 65 FF 23 78 AD D6 71 8B 7C 7C 19 61 F0 99 1B
           :     84 24 43 77 21 52 C9 E0 AD
129 02   21:   INTEGER
           :     00 E9 5E 4A 5F 73 70 59 DC 60 DF 59 91 D4 50 29
           :     40 9E 60 FC 09
152 02    1:   INTEGER 1
           :   }
```

Curve-ID: brainpoolP192r1

```
  0 30  176: SEQUENCE {
  3 02    1:   INTEGER 1
  6 30   36:   SEQUENCE {
  8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
 17 02   25:     INTEGER
           :       00 C3 02 F4 1D 93 2A 36 CD A7 A3 46 30 93 D1 8D
           :       B7 8F CE 47 6D E1 A8 62 97
           :     }
 44 30   52:   SEQUENCE {
 46 04   24:     OCTET STRING
           :       6A 91 17 40 76 B1 E0 E1 9C 39 C0 31 FE 86 85 C1
```

```
            :         CA E0 40 E5 C6 9A 28 EF
  72 04   24:       OCTET STRING
            :         46 9A 28 EF 7C 28 CC A3 DC 72 1D 04 4F 44 96 BC
            :         CA 7E F4 14 6F BF 25 C9
            :       }
  98 04   49:     OCTET STRING
            :       04 C0 A0 64 7E AA B6 A4 87 53 B0 33 C5 6C B0 F0
            :       90 0A 2F 5C 48 53 37 5F D6 14 B6 90 86 6A BD 5B
            :       B8 8B 5F 48 28 C1 49 00 02 E6 77 3F A2 FA 29 9B
            :       8F
 149 02   25:     INTEGER
            :       00 C3 02 F4 1D 93 2A 36 CD A7 A3 46 2F 9E 9E 91
            :       6B 5B E8 F1 02 9A C4 AC C1
 176 02    1:     INTEGER 1
            :     }
```

Curve-ID: brainpoolP192t1

```
   0 30  176: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   36:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   25:     INTEGER
            :       00 C3 02 F4 1D 93 2A 36 CD A7 A3 46 30 93 D1 8D
            :       B7 8F CE 47 6D E1 A8 62 97
            :     }
  44 30   52:   SEQUENCE {
  46 04   24:     OCTET STRING
            :       C3 02 F4 1D 93 2A 36 CD A7 A3 46 30 93 D1 8D B7
            :       8F CE 47 6D E1 A8 62 94
  72 04   24:     OCTET STRING
            :       13 D5 6F FA EC 78 68 1E 68 F9 DE B4 3B 35 BE C2
            :       FB 68 54 2E 27 89 7B 79
            :     }
  98 04   49:   OCTET STRING
            :     04 3A E9 E5 8C 82 F6 3C 30 28 2E 1F E7 BB F4 3F
            :     A7 2C 44 6A F6 F4 61 81 29 09 7E 2C 56 67 C2 22
            :     3A 90 2A B5 CA 44 9D 00 84 B7 E5 B3 DE 7C CC 01
            :     C9
 149 02   25:   INTEGER
            :     00 C3 02 F4 1D 93 2A 36 CD A7 A3 46 2F 9E 9E 91
            :     6B 5B E8 F1 02 9A C4 AC C1
 176 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP224r1

```
   0 30  200: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   40:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   29:     INTEGER
            :       00 D7 C1 34 AA 26 43 66 86 2A 18 30 25 75 D1 D7
            :       87 B0 9F 07 57 97 DA 89 F5 7E C8 C0 FF
            :     }
  48 30   60:   SEQUENCE {
  50 04   28:     OCTET STRING
            :       68 A5 E6 2C A9 CE 6C 1C 29 98 03 A6 C1 53 0B 51
            :       4E 18 2A D8 B0 04 2A 59 CA D2 9F 43
  80 04   28:     OCTET STRING
            :       25 80 F6 3C CF E4 41 38 87 07 13 B1 A9 23 69 E3
            :       3E 21 35 D2 66 DB B3 72 38 6C 40 0B
            :     }
```

```
 110 04   57:   OCTET STRING
            :     04 0D 90 29 AD 2C 7E 5C F4 34 08 23 B2 A8 7D C6
            :     8C 9E 4C E3 17 4C 1E 6E FD EE 12 C0 7D 58 AA 56
            :     F7 72 C0 72 6F 24 C6 B8 9E 4E CD AC 24 35 4B 9E
            :     99 CA A3 F6 D3 76 14 02 CD
 169 02   29:   INTEGER
            :     00 D7 C1 34 AA 26 43 66 86 2A 18 30 25 75 D0 FB
            :     98 D1 16 BC 4B 6D DE BC A3 A5 A7 93 9F
 200 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP224t1

```
   0 30  200: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   40:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   29:     INTEGER
            :       00 D7 C1 34 AA 26 43 66 86 2A 18 30 25 75 D1 D7
            :       87 B0 9F 07 57 97 DA 89 F5 7E C8 C0 FF
            :     }
  48 30   60:   SEQUENCE {
  50 04   28:     OCTET STRING
            :       D7 C1 34 AA 26 43 66 86 2A 18 30 25 75 D1 D7 87
            :       B0 9F 07 57 97 DA 89 F5 7E C8 C0 FC
  80 04   28:     OCTET STRING
            :       4B 33 7D 93 41 04 CD 7B EF 27 1B F6 0C ED 1E D2
            :       0D A1 4C 08 B3 BB 64 F1 8A 60 88 8D
            :     }
 110 04   57:   OCTET STRING
            :     04 6A B1 E3 44 CE 25 FF 38 96 42 4E 7F FE 14 76
            :     2E CB 49 F8 92 8A C0 C7 60 29 B4 D5 80 03 74 E9
            :     F5 14 3E 56 8C D2 3F 3F 4D 7C 0D 4B 1E 41 C8 CC
            :     0D 1C 6A BD 5F 1A 46 DB 4C
 169 02   29:   INTEGER
            :     00 D7 C1 34 AA 26 43 66 86 2A 18 30 25 75 D0 FB
            :     98 D1 16 BC 4B 6D DE BC A3 A5 A7 93 9F
 200 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP256r1

```
   0 30  224: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   44:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   33:     INTEGER
            :       00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D
            :       72 6E 3B F6 23 D5 26 20 28 20 13 48 1D 1F 6E 53
            :       77
            :     }
  52 30   68:   SEQUENCE {
  54 04   32:     OCTET STRING
            :       7D 5A 09 75 FC 2C 30 57 EE F6 75 30 41 7A FF E7
            :       FB 80 55 C1 26 DC 5C 6C E9 4A 4B 44 F3 30 B5 D9
  88 04   32:     OCTET STRING
            :       26 DC 5C 6C E9 4A 4B 44 F3 30 B5 D9 BB D7 7C BF
            :       95 84 16 29 5C F7 E1 CE 6B CC DC 18 FF 8C 07 B6
            :     }
 122 04   65:   OCTET STRING
            :     04 8B D2 AE B9 CB 7E 57 CB 2C 4B 48 2F FC 81 B7
            :     AF B9 DE 27 E1 E3 BD 23 C2 3A 44 53 BD 9A CE 32
            :     62 54 7E F8 35 C3 DA C4 FD 97 F8 46 1A 14 61 1D
```

```
            :     C9 C2 77 45 13 2D ED 8E 54 5C 1D 54 C7 2F 04 69
            :     97
 189 02   33:   INTEGER
            :     00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D
            :     71 8C 39 7A A3 B5 61 A6 F7 90 1E 0E 82 97 48 56
            :     A7
 224 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP256t1

```
   0 30  224: SEQUENCE {
   3 02    1:   INTEGER 1
   6 30   44:   SEQUENCE {
   8 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  17 02   33:     INTEGER
            :       00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D
            :       72 6E 3B F6 23 D5 26 20 28 20 13 48 1D 1F 6E 53
            :       77
            :     }
  52 30   68:   SEQUENCE {
  54 04   32:     OCTET STRING
            :       A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D 72
            :       6E 3B F6 23 D5 26 20 28 20 13 48 1D 1F 6E 53 74
  88 04   32:     OCTET STRING
            :       66 2C 61 C4 30 D8 4E A4 FE 66 A7 73 3D 0B 76 B7
            :       BF 93 EB C4 AF 2F 49 25 6A E5 81 01 FE E9 2B 04
            :     }
 122 04   65:   OCTET STRING
            :     04 A3 E8 EB 3C C1 CF E7 B7 73 22 13 B2 3A 65 61
            :     49 AF A1 42 C4 7A AF BC 2B 79 A1 91 56 2E 13 05
            :     F4 2D 99 6C 82 34 39 C5 6D 7F 7B 22 E1 46 44 41
            :     7E 69 BC B6 DE 39 D0 27 00 1D AB E8 F3 5B 25 C9
            :     BE
 189 02   33:   INTEGER
            :     00 A9 FB 57 DB A1 EE A9 BC 3E 66 0A 90 9D 83 8D
            :     71 8C 39 7A A3 B5 61 A6 F7 90 1E 0E 82 97 48 56
            :     A7
 224 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP320r1

```
   0 30  272: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   52:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   41:     INTEGER
            :       00 D3 5E 47 20 36 BC 4F B7 E1 3C 78 5E D2 01 E0
            :       65 F9 8F CF A6 F6 F4 0D EF 4F 92 B9 EC 78 93 EC
            :       28 FC D4 12 B1 F1 B3 2E 27
            :     }
  61 30   84:   SEQUENCE {
  63 04   40:     OCTET STRING
            :       3E E3 0B 56 8F BA B0 F8 83 CC EB D4 6D 3F 3B B8
            :       A2 A7 35 13 F5 EB 79 DA 66 19 0E B0 85 FF A9 F4
            :       92 F3 75 A9 7D 86 0E B4
 105 04   40:     OCTET STRING
            :       52 08 83 94 9D FD BC 42 D3 AD 19 86 40 68 8A 6F
            :       E1 3F 41 34 95 54 B4 9A CC 31 DC CD 88 45 39 81
            :       6F 5E B4 AC 8F B1 F1 A6
            :     }
 147 04   81:   OCTET STRING
```

```
             :     04 43 BD 7E 9A FB 53 D8 B8 52 89 BC C4 8E E5 BF
             :     E6 F2 01 37 D1 0A 08 7E B6 E7 87 1E 2A 10 A5 99
             :     C7 10 AF 8D 0D 39 E2 06 11 14 FD D0 55 45 EC 1C
             :     C8 AB 40 93 24 7F 77 27 5E 07 43 FF ED 11 71 82
             :     EA A9 C7 78 77 AA AC 6A C7 D3 52 45 D1 69 2E 8E
             :     E1
 230 02   41:   INTEGER
             :     00 D3 5E 47 20 36 BC 4F B7 E1 3C 78 5E D2 01 E0
             :     65 F9 8F CF A5 B6 8F 12 A3 2D 48 2E C7 EE 86 58
             :     E9 86 91 55 5B 44 C5 93 11
 273 02    1:   INTEGER 1
             :   }
```

Curve-ID: brainpoolP320t1

```
   0 30  272: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   52:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   41:     INTEGER
             :       00 D3 5E 47 20 36 BC 4F B7 E1 3C 78 5E D2 01 E0
             :       65 F9 8F CF A6 F6 F4 0D EF 4F 92 B9 EC 78 93 EC
             :       28 FC D4 12 B1 F1 B3 2E 27
             :     }
  61 30   84:   SEQUENCE {
  63 04   40:     OCTET STRING
             :       D3 5E 47 20 36 BC 4F B7 E1 3C 78 5E D2 01 E0 65
             :       F9 8F CF A6 F6 F4 0D EF 4F 92 B9 EC 78 93 EC 28
             :       FC D4 12 B1 F1 B3 2E 24
 105 04   40:     OCTET STRING
             :       A7 F5 61 E0 38 EB 1E D5 60 B3 D1 47 DB 78 20 13
             :       06 4C 19 F2 7E D2 7C 67 80 AA F7 7F B8 A5 47 CE
             :       B5 B4 FE F4 22 34 03 53
             :     }
 147 04   81:   OCTET STRING
             :     04 92 5B E9 FB 01 AF C6 FB 4D 3E 7D 49 90 01 0F
             :     81 34 08 AB 10 6C 4F 09 CB 7E E0 78 68 CC 13 6F
             :     FF 33 57 F6 24 A2 1B ED 52 63 BA 3A 7A 27 48 3E
             :     BF 66 71 DB EF 7A BB 30 EB EE 08 4E 58 A0 B0 77
             :     AD 42 A5 A0 98 9D 1E E7 1B 1B 9B C0 45 5F B0 D2
             :     C3
 230 02   41:   INTEGER
             :     00 D3 5E 47 20 36 BC 4F B7 E1 3C 78 5E D2 01 E0
             :     65 F9 8F CF A5 B6 8F 12 A3 2D 48 2E C7 EE 86 58
             :     E9 86 91 55 5B 44 C5 93 11
 273 02    1:   INTEGER 1
             :   }
```

Curve-ID: brainpoolP384r1

```
   0 30  320: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   60:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   49:     INTEGER
             :       00 8C B9 1E 82 A3 38 6D 28 0F 5D 6F 7E 50 E6 41
             :       DF 15 2F 71 09 ED 54 56 B4 12 B1 DA 19 7F B7 11
             :       23 AC D3 A7 29 90 1D 1A 71 87 47 00 13 31 07 EC
             :       53
             :     }
  69 30  100:   SEQUENCE {
  71 04   48:     OCTET STRING
             :       7B C3 82 C6 3D 8C 15 0C 3C 72 08 0A CE 05 AF A0
```

```
            :       C2 BE A2 8E 4F B2 27 87 13 91 65 EF BA 91 F9 0F
            :       8A A5 81 4A 50 3A D4 EB 04 A8 C7 DD 22 CE 28 26
 121 04   48:     OCTET STRING
            :       04 A8 C7 DD 22 CE 28 26 8B 39 B5 54 16 F0 44 7C
            :       2F B7 7D E1 07 DC D2 A6 2E 88 0E A5 3E EB 62 D5
            :       7C B4 39 02 95 DB C9 94 3A B7 86 96 FA 50 4C 11
            :     }
 171 04   97:   OCTET STRING
            :     04 1D 1C 64 F0 68 CF 45 FF A2 A6 3A 81 B7 C1 3F
            :     6B 88 47 A3 E7 7E F1 4F E3 DB 7F CA FE 0C BD 10
            :     E8 E8 26 E0 34 36 D6 46 AA EF 87 B2 E2 47 D4 AF
            :     1E 8A BE 1D 75 20 F9 C2 A4 5C B1 EB 8E 95 CF D5
            :     52 62 B7 0B 29 FE EC 58 64 E1 9C 05 4F F9 91 29
            :     28 0E 46 46 21 77 91 81 11 42 82 03 41 26 3C 53
            :     15
 270 02   49:   INTEGER
            :     00 8C B9 1E 82 A3 38 6D 28 0F 5D 6F 7E 50 E6 41
            :     DF 15 2F 71 09 ED 54 56 B3 1F 16 6E 6C AC 04 25
            :     A7 CF 3A B6 AF 6B 7F C3 10 3B 88 32 02 E9 04 65
            :     65
 321 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP384t1

```
   0 30  320: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   60:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   49:     INTEGER
            :       00 8C B9 1E 82 A3 38 6D 28 0F 5D 6F 7E 50 E6 41
            :       DF 15 2F 71 09 ED 54 56 B4 12 B1 DA 19 7F B7 11
            :       23 AC D3 A7 29 90 1D 1A 71 87 47 00 13 31 07 EC
            :       53
            :     }
  69 30  100:   SEQUENCE {
  71 04   48:     OCTET STRING
            :       8C B9 1E 82 A3 38 6D 28 0F 5D 6F 7E 50 E6 41 DF
            :       15 2F 71 09 ED 54 56 B4 12 B1 DA 19 7F B7 11 23
            :       AC D3 A7 29 90 1D 1A 71 87 47 00 13 31 07 EC 50
 121 04   48:     OCTET STRING
            :       7F 51 9E AD A7 BD A8 1B D8 26 DB A6 47 91 0F 8C
            :       4B 93 46 ED 8C CD C6 4E 4B 1A BD 11 75 6D CE 1D
            :       20 74 AA 26 3B 88 80 5C ED 70 35 5A 33 B4 71 EE
            :     }
 171 04   97:   OCTET STRING
            :     04 18 DE 98 B0 2D B9 A3 06 F2 AF CD 72 35 F7 2A
            :     81 9B 80 AB 12 EB D6 53 17 24 76 FE CD 46 2A AB
            :     FF C4 FF 19 1B 94 6A 5F 54 D8 D0 AA 2F 41 88 08
            :     CC 25 AB 05 69 62 D3 06 51 A1 14 AF D2 75 5A D3
            :     36 74 7F 93 47 5B 7A 1F CA 3B 88 F2 B6 A2 08 CC
            :     FE 46 94 08 58 4D C2 B2 91 26 75 BF 5B 9E 58 29
            :     28
 270 02   49:   INTEGER
            :     00 8C B9 1E 82 A3 38 6D 28 0F 5D 6F 7E 50 E6 41
            :     DF 15 2F 71 09 ED 54 56 B3 1F 16 6E 6C AC 04 25
            :     A7 CF 3A B6 AF 6B 7F C3 10 3B 88 32 02 E9 04 65
            :     65
 321 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP512r1

```
   0 30  418: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   76:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   65:     INTEGER
            :       00 AA DD 9D B8 DB E9 C4 8B 3F D4 E6 AE 33 C9 FC
            :       07 CB 30 8D B3 B3 C9 D2 0E D6 63 9C CA 70 33 08
            :       71 7D 4D 9B 00 9B C6 68 42 AE CD A1 2A E6 A3 80
            :       E6 28 81 FF 2F 2D 82 C6 85 28 AA 60 56 58 3A 48
            :       F3
            :     }
  85 30  132:   SEQUENCE {
  88 04   64:     OCTET STRING
            :       78 30 A3 31 8B 60 3B 89 E2 32 71 45 AC 23 4C C5
            :       94 CB DD 8D 3D F9 16 10 A8 34 41 CA EA 98 63 BC
            :       2D ED 5D 5A A8 25 3A A1 0A 2E F1 C9 8B 9A C8 B5
            :       7F 11 17 A7 2B F2 C7 B9 E7 C1 AC 4D 77 FC 94 CA
 154 04   64:     OCTET STRING
            :       3D F9 16 10 A8 34 41 CA EA 98 63 BC 2D ED 5D 5A
            :       A8 25 3A A1 0A 2E F1 C9 8B 9A C8 B5 7F 11 17 A7
            :       2B F2 C7 B9 E7 C1 AC 4D 77 FC 94 CA DC 08 3E 67
            :       98 40 50 B7 5E BA E5 DD 28 09 BD 63 80 16 F7 23
            :     }
 220 04  129:   OCTET STRING
            :     04 81 AE E4 BD D8 2E D9 64 5A 21 32 2E 9C 4C 6A
            :     93 85 ED 9F 70 B5 D9 16 C1 B4 3B 62 EE F4 D0 09
            :     8E FF 3B 1F 78 E2 D0 D4 8D 50 D1 68 7B 93 B9 7D
            :     5F 7C 6D 50 47 40 6A 5E 68 8B 35 22 09 BC B9 F8
            :     22 7D DE 38 5D 56 63 32 EC C0 EA BF A9 CF 78 22
            :     FD F2 09 F7 00 24 A5 7B 1A A0 00 C5 5B 88 1F 81
            :     11 B2 DC DE 49 4A 5F 48 5E 5B CA 4B D8 8A 27 63
            :     AE D1 CA 2B 2F A8 F0 54 06 78 CD 1E 0F 3A D8 08
            :     92
 352 02   65:   INTEGER
            :     00 AA DD 9D B8 DB E9 C4 8B 3F D4 E6 AE 33 C9 FC
            :     07 CB 30 8D B3 B3 C9 D2 0E D6 63 9C CA 70 33 08
            :     70 55 3E 5C 41 4C A9 26 19 41 86 61 19 7F AC 10
            :     47 1D B1 D3 81 08 5D DA DD B5 87 96 82 9C A9 00
            :     69
 419 02    1:   INTEGER 1
            :   }
```

Curve-ID: brainpoolP512t1

```
   0 30  418: SEQUENCE {
   4 02    1:   INTEGER 1
   7 30   76:   SEQUENCE {
   9 06    7:     OBJECT IDENTIFIER prime-field (1 2 840 10045 1 1)
  18 02   65:     INTEGER
            :       00 AA DD 9D B8 DB E9 C4 8B 3F D4 E6 AE 33 C9 FC
            :       07 CB 30 8D B3 B3 C9 D2 0E D6 63 9C CA 70 33 08
            :       71 7D 4D 9B 00 9B C6 68 42 AE CD A1 2A E6 A3 80
            :       E6 28 81 FF 2F 2D 82 C6 85 28 AA 60 56 58 3A 48
            :       F3
            :     }
  85 30  132:   SEQUENCE {
  88 04   64:     OCTET STRING
            :       AA DD 9D B8 DB E9 C4 8B 3F D4 E6 AE 33 C9 FC 07
            :       CB 30 8D B3 B3 C9 D2 0E D6 63 9C CA 70 33 08 71
            :       7D 4D 9B 00 9B C6 68 42 AE CD A1 2A E6 A3 80 E6
            :       28 81 FF 2F 2D 82 C6 85 28 AA 60 56 58 3A 48 F0
 154 04   64:     OCTET STRING
            :       7C BB BC F9 44 1C FA B7 6E 18 90 E4 68 84 EA E3
```

```
            :       21 F7 0C 0B CB 49 81 52 78 97 50 4B EC 3E 36 A6
            :       2B CD FA 23 04 97 65 40 F6 45 00 85 F2 DA E1 45
            :       C2 25 53 B4 65 76 36 89 18 0E A2 57 18 67 42 3E
            :     }
220 04  129:   OCTET STRING
            :     04 64 0E CE 5C 12 78 87 17 B9 C1 BA 06 CB C2 A6
            :     FE BA 85 84 24 58 C5 6D DE 9D B1 75 8D 39 C0 31
            :     3D 82 BA 51 73 5C DB 3E A4 99 AA 77 A7 D6 94 3A
            :     64 F7 A3 F2 5F E2 6F 06 B5 1B AA 26 96 FA 90 35
            :     DA 5B 53 4B D5 95 F5 AF 0F A2 C8 92 37 6C 84 AC
            :     E1 BB 4E 30 19 B7 16 34 C0 11 31 15 9C AE 03 CE
            :     E9 D9 93 21 84 BE EF 21 6B D7 1D F2 DA DF 86 A6
            :     27 30 6E CF F9 6D BB 8B AC E1 98 B6 1E 00 F8 B3
            :     32
352 02   65:   INTEGER
            :     00 AA DD 9D B8 DB E9 C4 8B 3F D4 E6 AE 33 C9 FC
            :     07 CB 30 8D B3 B3 C9 D2 0E D6 63 9C CA 70 33 08
            :     70 55 3E 5C 41 4C A9 26 19 41 86 61 19 7F AC 10
            :     47 1D B1 D3 81 08 5D DA DD B5 87 96 82 9C A9 00
            :     69
419 02    1:   INTEGER 1
            :   }
```

Note. Binaries of the encoded domain parameters are available on the webpages [BP] of the ECC-Brainpool. The Binaries were processed using Peter Gutman's dumpasn1 utility to generate the output. The source-code for the dumpasn1 utility is available at [DUM].

Editor's address:
Dr. Manfred Lochter
BSI
Postfach 200363
53133 Bonn

email: manfred.lochter@bsi.bund.de
Tel.: +49 1888 9582 643
Fax: +49 1888 9582 90 643