Elliptic Curve Cryptography (ECC) Brainpool Standard
Curves and Curve Generation

Abstract

This memo proposes several elliptic curve domain parameters over finite prime fields for use in cryptographic applications. The domain parameters are consistent with the relevant international standards, and can be used in X.509 certificates and certificate revocation lists (CRLs), for Internet Key Exchange (IKE), Transport Layer Security (TLS), XML signatures, and all applications or protocols based on the cryptographic message syntax (CMS).

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5639.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
Table of Contents

1. Introduction .................................................... 3
   1.1. Scope and Relation to Other Specifications ............... 4
   1.2. Requirements Language ....................................... 4

2. Requirements on the Elliptic Curve Domain Parameters .......... 4
   2.1. Security Requirements ....................................... 5
   2.2. Technical Requirements ..................................... 6

3. Domain Parameter Specification .................................. 8
   3.1. Domain Parameters for 160-Bit Curves .................... 8
   3.2. Domain Parameters for 192-Bit Curves .................... 9
   3.3. Domain Parameters for 224-Bit Curves ................... 10
   3.4. Domain Parameters for 256-Bit Curves ................... 11
   3.5. Domain Parameters for 320-Bit Curves ................... 12
   3.6. Domain Parameters for 384-Bit Curves ................... 13
   3.7. Domain Parameters for 512-Bit Curves ................... 14

4. Object Identifiers and ASN.1 Syntax ............................ 15
   4.1. Object Identifiers ......................................... 15
   4.2. ASN.1 Syntax for Usage with X.509 Certificates .......... 16

5. Security Considerations ........................................ 17

6. Intellectual Property Rights .................................... 18

7. References .................................................... 18
   7.1. Normative References ....................................... 18
   7.2. Informative References ..................................... 19

Appendix A. Pseudo-Random Generation of Parameters .............. 22
   A.1. Generation of Prime Numbers ............................... 22
   A.2. Generation of Pseudo-Random Curves ..................... 24
1. Introduction

Although several standards for elliptic curves and domain parameters exist (e.g., [ANSI1], [FIPS], or [SEC2]), some major issues have still not been addressed:

- Not all parameters have been generated in a verifiably pseudo-random way. In particular, the seeds from which the curve parameters were derived have been chosen ad hoc, leaving out an essential part of the security proof.

- The primes selected for the base fields have a very special form facilitating efficient implementation. This does not only contradict the approach of pseudo-random parameters, but also increases the risk of implementations violating one of the numerous patents for fast modular arithmetic with special primes.

- No proofs are provided that the proposed parameters do not belong to those classes of parameters that are susceptible to cryptanalytic attacks with sub-exponential complexity.

- Recent research results seem to indicate a potential for new attacks on elliptic curve cryptosystems. At least for applications with the highest security demands or under circumstances that complicate a change of parameters in response to new attacks, the inclusion of a corresponding security requirement for domain parameters (the class group condition, see Section 2) is justified.

- Some of the proposed subgroups have a non-trivial cofactor, which demands additional checks by cryptographic applications to prevent small subgroup attacks (see [ANSI1] or [SEC1]).

- The domain parameters specified do not cover all bit lengths that correspond to the commonly used key lengths for symmetric cryptographic algorithms. In particular, there is no 512-bit curve defined, but only one with a 521-bit length, which may be disadvantageous for some implementations.

Furthermore, many of the parameters specified by the existing standards are identical (see [SEC2] for a comparison). Thus, there is still a need for additional elliptic curve domain parameters that overcome the above limitations.
1.1. Scope and Relation to Other Specifications

This RFC specifies elliptic curve domain parameters over prime fields \( GF(p) \) with \( p \) having a length of 160, 192, 224, 256, 320, 384, and 512 bits. These parameters were generated in a pseudo-random, yet completely systematic and reproducible, way and have been verified to resist current cryptanalytic approaches. The parameters are compliant with ANSI X9.62 [ANSI1] and ANSI X9.63 [ANSI2], ISO/IEC 14888 [ISO1] and ISO/IEC 15946 [ISO2], ETSI TS 102 176-1 [ETSI], as well as with FIPS-186-2 [FIPS], and the Efficient Cryptography Group (SECG) specifications ([SEC1] and [SEC2]).

Furthermore, this document identifies the security and implementation requirements for the parameters, and describes the methods used for the pseudo-random generation of the parameters.

Finally, this RFC defines ASN.1 object identifiers for all elliptic curve domain parameter sets specified herein, e.g., for use in X.509 certificates.

This document does neither address the cryptographic algorithms to be used with the specified parameters nor their application in other standards. However, it is consistent with the following RFCs that specify the usage of elliptic curve cryptography in protocols and applications:

- [RFC5753] for the cryptographic message syntax (CMS)
- [RFC3279] and [RFC5480] for X.509 certificates and CRLs
- [RFC4050] for XML signatures
- [RFC4492] for TLS
- [RFC4754] for IKE

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Requirements on the Elliptic Curve Domain Parameters

Throughout this memo, let \( p > 3 \) be a prime and \( GF(p) \) a finite field (sometimes also referred to as Galois Field or GF(p)) with \( p \) elements. For given \( A \) and \( B \) with non-zero \( 4A^3 + 27B^2 \mod p \), the set of solutions \((x,y)\) for the equation \( E: y^2 = x^3 + A*x + B \mod p \)
over GF(p) together with a neutral element 0 and well-defined laws
for addition and inversion define a group E(GF(p)) -- the group of
GF(p) rational points on E. Typically, for cryptographic
applications, an element G of prime order q is chosen in E(GF(p)).

A comprehensive introduction to elliptic curve cryptography can be
found in [CFDA] and [BSS].

Note 1: We choose {0,...,p-1} as a set of representatives for the
elements of GF(p). This choice induces a natural ordering on GF(p).

2.1. Security Requirements

The following security requirements are either motivated by known
cryptographic analysis or aim to enhance trust in the recommended
curves. As this specification aims at a particularly high level of
security, a restrictive position is taken here. Nevertheless, it may
be sensible to slightly deviate from these requirements for certain
applications (e.g., in order to achieve higher computational
performance). More details on requirements for cryptographically
strong elliptic curves can be found in [CFDA] and [BSS].

1. Immunity to attacks using the Weil or Tate Pairing. These
attacks allow the embedding of the cyclic subgroup generated by G
into the group of units of a degree-l extension GF(p^l) of GF(p),
where sub-exponential attacks on the discrete logarithm problem
(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 require (q-1)/l < 100, which means that l is close to
the maximum possible value. This requirement is considerably
stronger than those of [SEC2] and [ANSI2] and also excludes
supersingular curves, as those are the curves of order p+1.

2. The trace is not equal to one. 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 elliptic curve discrete logarithm problem
( ECDLP) on trace one curves. Note that these curves are also
excluded by requirement 5 of Section 2.2.

3. Large class number. The class number of the maximal order of the
quotient field of the endomorphism ring End(E) of E is larger
than 10^7. Generally, 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). Although there are no efficient attacks exploiting a
small class number, recent work ([JMV] and [HR]) also may be seen
as argument for the class number condition.
4. Prime group order. The group order \#E(GF(p)) shall be a prime number in order to counter small-subgroup attacks (see [HMV]). Therefore, all groups proposed in this RFC have cofactor 1. Note that curves with prime order have no point of order 2 and therefore no point with y-coordinate 0.

5. Verifiably pseudo-random. The elliptic curve domain parameters shall be generated in a pseudo-random manner using seeds that are generated in a systematic and comprehensive way. The methods by which the parameters have been obtained are explained in Appendix A.

6. Proof of security. For all curves, a proof should be given that all security requirements are met. These proofs are provided in [EBP].

In [BG], attacks are described that apply to elliptic curve domain parameters where q-1 has a factor u in the order of q^(1/3). However, the circumstances under which these attacks are applicable can be avoided in most applications. Therefore, no corresponding security requirement is stated here. However, it is highly recommended that developers verify the security of their implementations against this kind of attack.

2.2. Technical Requirements

Commercial demands and experience with existing implementations lead to the following technical requirements for the elliptic curve domain parameters.

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 that are appropriate for the underlying symmetric algorithms. The existing standards 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. For p = 3 mod 4, we get (y^2)^((p+1)/4) = y*y^((p-1)/2), which is either y or -y by Fermat’s Little Theorem; hence, y can be computed very efficiently using the curve equation. This requirement is not always met by the parameters defined in existing standards.
3. The curves shall be GF(p)-isomorphic to a curve E': $y^2 = x^3 + A'x + B' \mod p$ with $A' = -3 \mod p$. This property permits the use of the arithmetical advantages of curves with $A = -3$, as shown by Brier and Joyce [BJ]. For $p = 3 \mod 4$, approximately half of the isomorphism classes of elliptic curves over GF(p) contain a curve E' with $A' = -3 \mod p$. Precisely, if a curve is given by $E: y^2 = x^3 + Ax + B \mod p$ with $-3 = A*u^4$ being solvable in GF(p) and $u=Z$ is a solution to this equation, then the requirement is fulfilled by means of the quadratic twist $E': y^2 = x^3 + Z^4*A*x + Z^6*B \mod p$, and the GF(p)-isomorphism is given by $F(x,y) := (x*Z^2, y*Z^3)$. Due to this isomorphism, $E(GF(p))$ and $E'(GF(p))$ have the same number of points, share the same algebraic structure, and hence offer the same level of security. This constraint has also been used by [SEC2] and [FIPS].

4. The prime p must not be of any special form; this requirement is met by a verifiably pseudo-random generation of the parameters (see requirement 5 in Section 2.1). Although parameters specified by existing standards do not meet this requirement, the need for such curves over (pseudo-)randomly chosen fields has already been foreseen by the Standards for Efficient Cryptography Group (SECG), see [SEC2].

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 order to thwart attacks on digital signature schemes, some authors propose to use $q > p$, but the attacks described, e.g., in [BRS], appear infeasible in a well-designed Public Key Infrastructure (PKI).

6. B shall be a non-square mod p. Otherwise, the compressed representations of the curve-points (0,0) and (0,X), with X being the square root of B with a least significant bit of 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 the attack described in [G] impossible. It can therefore also be seen as a security requirement. This constraint has not been specified by existing standards.
3. Domain Parameter Specification

In this section, the elliptic curve domain parameters proposed are specified in the following way.

For all curves, an ID is given by which it can be referenced.

p is the prime specifying the base field.

A and B are the coefficients of the equation \( y^2 = x^3 + Ax + B \mod p \) defining the elliptic curve.

G = (x,y) is the base point, i.e., a point in E of prime order, with x and y being its x- and y-coordinates, respectively.

q is the prime order of the group generated by G.

h is the cofactor of G in E, i.e., \( \#E(GF(p))/q \).

For the twisted curve, we also give the coefficient Z that defines the isomorphism F (see requirement 3 in Section 2.2).

The methods for the generation of the parameters are given in Appendix A. Proofs for the fulfillment of the security requirements specified in Section 2.1 are given in [EBP].

3.1. Domain Parameters for 160-Bit Curves

Curve-ID: brainpoolP160r1

\[
\begin{align*}
 p &= E95E4A5F737059DC60DFC7AD95B3D8139515620F \\
 A &= 340E7BE2A280EB74E2BE61BADA745D97E8F7C300 \\
 B &= 1E589A8595423412134FAA2DBDEC95C8D8675E58 \\
 x &= BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3 \\
 y &= 1667CB477A1A8EC338F94741669C976316DA6321 \\
 q &= E95E4A5F737059DC60DF5991D45029409E60FC09 \\
 h &= 1
\end{align*}
\]
#Twisted curve

Curve-ID: brainpoolP160t1

\[
\begin{align*}
Z &= 24DBFF5DEC9B986BBFE5295A29BFBAE45E0F5D0B \\
A &= E95E4A5F737059DC60DFC7AD95B3D8139515620C \\
B &= 7A556B6DAE535B7B51ED2C4D7DA8A7A0B5C55F380 \\
x &= B199B13B9B34EFC1397E64BAEB05ACC265FF2378 \\
y &= ADD6718B7C7C1961F0991B842443772152C9E0AD \\
q &= E95E4A5F737059DC60DF5991D45029409E60FC09 \\
h &= 1
\end{align*}
\]

3.2. Domain Parameters for 192-Bit Curves

Curve-ID: brainpoolP192r1

\[
\begin{align*}
p &= C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297 \\
A &= 6A91174076B1E0E19C39C031FE8685C1CAE040E5C69A28EF \\
B &= 469A28EF7C28CCA3DC721D044F4496BCCA7EF4146FBE5C9 \\
x &= C0A0647EAAB6A48753B033C56CB0F0900A2F5C4853375FD6 \\
y &= 14B690866ABD5BB88B5F4828C1490002E6773FA2FA299B8F \\
q &= C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1 \\
h &= 1
\end{align*}
\]

#Twisted curve

Curve-ID: brainpoolP192t1

\[
\begin{align*}
Z &= 1B6F5CC8DB4D7A19458A9CB80DC2295E5EB9C3732104CB \\
A &= C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86294 \\
B &= 13D56FBAEC78681E68F9DEB43B35BEC2FB68542E27897B79 \\
x &= 3AE9E58C82F63C30282E1FE7BBF43FA72C446AF6F4618129
\end{align*}
\]
3.3. Domain Parameters for 224-Bit Curves

Curve-ID: brainpoolP224r1

\[ p = D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF \]
\[ A = 68A5E62CA9CE6C1C299803A6C1530B514E182AD8B0042A59CAD29F43 \]
\[ B = 2580F63CCFE44138870713B1A92369E33E2135D266DBB3723B6C400B \]
\[ x = 0D9029AD2C7E5CF4340823B2A87DC68C9E4CE3174C1E6EFDEE12C07D \]
\[ y = 58AA56F772C0726F24C6B89E4ECDAC24354B9E99CAA3F6D3761402CD \]
\[ q = D7C134AA264366862A18302575D0FB98D116BC4B6DEBEC4A5A7939F \]
\[ h = 1 \]

Twisted curve

Curve-ID: brainpoolP224t1

\[ Z = 2DF271E14427A346910CF7A2E6CFA7B3F484E5C2CCCE1C8B730E28B3F \]
\[ A = D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FC \]
\[ B = 4B337D934104CD7BEF271BF60CED1ED20DA14C08B3BB64F18A6088D \]
\[ x = 6AB1E344CE25FF3896424E7FFE14762ECD49F8928AC0C76029B4D5B0 \]
\[ y = 0374E9F5143E568CD23F3F4D7C0D4B1E41C8CC0D1C6ABD5F1A46DB4C \]
\[ q = D7C134AA264366862A18302575D0FB98D116BC4B6DEBEC4A5A7939F \]
\[ h = 1 \]
3.4. Domain Parameters for 256-Bit Curves

Curve-ID: brainpoolP256r1

\[
\begin{align*}
p &= A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377 \\
A &= 7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9 \\
B &= 26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18F8C07B6 \\
x &= 8BD2AEB9CB7E57CB2C4B482FFC8187A9F9DE27E1E3BD23C23A4453B9ACE3262 \\
y &= 547EF835C3DAC4FD97F8461A14611DC9C27745132DED8E545C1D54C72F046997 \\
q &= A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7 \\
h &= 1
\end{align*}
\]
# Twisted curve

Curve-ID: brainpoolP256t1

\[
\begin{align*}
Z &= 3E2D4BD9597B58639AE7AA669CAB9837CF5CF20A2C852D10F655668DFC150EF0 \\
A &= A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5374 \\
B &= 662C61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04 \\
x &= A3E8EB3CC1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F4 \\
y &= 2D996C823439C56D7F7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE \\
q &= A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7 \\
h &= 1
\end{align*}
\]
3.5. Domain Parameters for 320-Bit Curves

**Curve-ID: brainpoolP320r1**

\[
\begin{align*}
p &= \text{D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27} \\
A &= \text{3EE30B568FBAB0F883CCEBD46D3F3BB8A2A73513F5EB79DA66190EB085FFA9F492F375A97D860EB4} \\
B &= \text{520B88394DFDBC42D3AD198640688A6FE13F41349554B49ACC31DCCD884539816F5EB4AC8FB1F1A6} \\
x &= \text{43BD7E9AFB53D8B85289BCC48EE5BFE6F20137D10A087EB6E7871E2A10A599C710AF80D39E20611} \\
y &= \text{14FDD05545EC1CC8AB4093247F77275E0743FFED117182EEAA9C77877AAAC6AC7D352451692E8EE1} \\
q &= \text{D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311} \\
h &= 1
\end{align*}
\]

#Twisted curve

**Curve-ID: brainpoolP320t1**

\[
\begin{align*}
Z &= \text{15F75CAF668077F7E85B42EB01F0A81FF56ED6191D55CB82B7D861458A18F} \\
A &= \text{D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E24} \\
B &= \text{A7F561E038EB1ED560B3D147DB782013064C19F27ED27C6780AAF77FB8A547CEB5B4F6F422340353} \\
x &= \text{925BE9FB01AF6C4F6B4D3E7D4990010F813408A106C4F09CB7EE07868CC136F} \\
y &= \text{63BA3A7A27483EBF6671DBEF7ABB30EBEE084E58A0B77AD42A50989D1EE7} \\
q &= \text{D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311} \\
h &= 1
\end{align*}
\]
3.6. Domain Parameters for 384-Bit Curves

Curve-ID: brainpoolP384r1

\[
p = \text{8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB711}
\text{23ACD3A729901D1A71874700133107EC53}
\]
\[
A = \text{7BC382C63D8C150C3C72080ACE05FAA0C2BEA28E4FB22787139165EFBA91F9}
\text{0F8AA5814A503AD4EB04A8C7DD22CE2826}
\]
\[
B = \text{04ABC7DD22CE2826B39B55416F0447C2FB77DE107DCD2A62E880EA53EEB62}
\text{D57CB4390295DBC9943AB78696FA504C11}
\]
\[
x = \text{1D1C64F068CF45FFA2A63A1B7C13F6B8847A3E77EF14FE3DB7FCAFE0CBD10}
\text{E8E826E03436D646AAEF87B2E247D4AF1E}
\]
\[
y = \text{8ABE1D75209F9C2A45CB1EB8E95CFD55262B70B29FECC5864E19C054FF99129}
\text{280E466217791811142820341263C5315}
\]
\[
q = \text{8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425}
\text{A7CF3AB6AF6B7FC3103B883202E9046565}
\]
\[
h = 1
\]

#Twisted curve

Curve-ID: brainpoolP384t1

\[
Z = \text{41DFE8DD399331F7166A66076734A89CD0D2BCDB7D68E44E1F378F41ECBAE}
\text{97D2B63B8C87BCCDCC5DA39BE859291C}
\]
\[
A = \text{8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB711}
\text{23ACD3A729901D1A71874700133107EC50}
\]
\[
B = \text{7F519EADA7BDA81BD826DBA647910F8C489346ED8CCDC6E4B1ABD11756DCE}
\text{1D2074AA263B88805CED70355A33B471EE}
\]
\[
x = \text{18DE98B02DB9A306F2AFCD7235F72A819B80AB12EBD653172476FECD462AAB}
\text{FFC4FF191B9465A5F548D0AA2F418808CC}
\]
\[
y = \text{25AB056962D30651A114AFD2755AD336747F93475B7A1FCA3B88F2B6A208CC}
\text{FE469408584DC2B2912675BF5B9E582928}
\]
\[
q = \text{8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425}
\text{A7CF3AB6AF6B7FC3103B883202E9046565}
\]
\[
h = 1
\]
3.7. Domain Parameters for 512-Bit Curves

Curve-ID: brainpoolP512r1

\[ \begin{align*}
p &= \text{AADD9DB8DE9C48B3FD4E6AE33C9FC07CB308DB3C9D20ED6639CCA703308} \\
A &= \text{7830A3318603B89E2327145AC234CC594CBDD8D3DF91610A83441CAEA9863} \\
B &= \text{3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117} \\
x &= \text{81AEE4BDD82ED9645A21322E9C46A9385ED9F005470B5D916C1B43B62EEF4D009} \\
y &= \text{7DE3E855663232ECC0EABFA9CF7822FDF209F70024A57B1AA000C55B881F81} \\
g &= \text{AADD9DB8DE9C48B3FD4E6AE33C9FC07CB308DB3C9D20ED6639CCA703308} \\
h &= 1
\end{align*} \]

# Twisted curve

Curve-ID: brainpoolP512t1

\[ \begin{align*}
Z &= \text{12EE58E6764838B69782136F0F023BA06E2769571605092E60A80BEB212B} \\
A &= \text{AADD9DB8DE9C48B3FD4E6AE33C9FC07CB308DB3C9D20ED6639CCA703308} \\
B &= \text{7CBBCF9441CFA7B6E1980E46884AE321F70C0BCB4981527897504BEC3E36} \\
x &= \text{640ECE5C12788717B9C126CB2CA6FEB858424585C6DED9BB175B0D39C03} \\
y &= \text{5B534BD595F5AF0FA2C892376C84ACE1BB4E3019B71634C01131159CAE03CE} \\
g &= \text{AADD9DB8DE9C48B3FD4E6AE33C9FC07CB308DB3C9D20ED6639CCA703308} \\
h &= 1
\end{align*} \]
4. Object Identifiers and ASN.1 Syntax

4.1. Object Identifiers

The root of the tree for the object identifiers defined in this specification is given by:

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

The object identifier ellipticCurve represents the tree for domain parameter sets. It has the following value:

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

The tree containing the object identifiers for each set of domain parameters defined in this RFC is:

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

The following object identifiers represent the domain parameter sets defined in this RFC:

```plaintext
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}

4.2. ASN.1 Syntax for Usage with X.509 Certificates

The domain parameters specified in this RFC SHALL be used with X.509 certificates in accordance with [RFC5480]. In particular,

- the algorithm field of subjectPublicKeyInfo MUST be set to:
  - id-ecPublicKey, if the algorithms that can be used with the subject public key are not restricted, or
  - id-ecDH to restrict the usage of the subject public key to Elliptic Curve Diffie-Hellman (ECDH) key agreement, or
  - id-ecMQV to restrict the usage of the subject public key to Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key agreement, and

- the field algorithm.parameter of subjectPublicKeyInfo MUST be of type:
  - namedCurve to specify the domain parameters by one of the Object Identifiers (OIDs) defined in Section 4.1, or
  - specifiedCurve to specify the domain parameters explicitly as defined in [RFC5480], or
  - implicitCurve, if the domain parameters are found in an issuer’s certificate.

If the domain parameters are explicitly specified using the type specifiedCurve in the field algorithm.parameter of subjectPublicKeyInfo, ANSI X9.62 [ANSI1] and [RFC5480] allow indicating whether or not a curve and base point have been generated verifiably in a pseudo-random way. Although the parameters specified in Section 3 have all been generated by the pseudo-random methods described in Appendix A, these algorithms deviate from those mandated in ANSI X9.62, A.3.3.1. Consequently, applications following ANSI X9.62 or [RFC5480] will not be able to verify the pseudo-randomness of the parameters. In order to avoid rejection of the parameters, the ASN.1 encoding SHOULD NOT specify that the curve or base point has been generated verifiably at random. In particular, certification authorities (CAs) SHOULD set the contents of specifiedCurve in the following way:

- version is set to ecpVer1(1).
5. Security Considerations

The level of security provided by symmetric ciphers and hash functions used in conjunction with the elliptic curve domain parameters specified in this RFC should roughly match or exceed the level provided by the domain parameters. The following table indicates the minimum key sizes for symmetric ciphers and hash functions providing at least (roughly) comparable security.
<table>
<thead>
<tr>
<th>elliptic curve domain parameters</th>
<th>minimum length of symmetric keys</th>
<th>hash functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>brainpoolP160r1</td>
<td>80</td>
<td>SHA-1, SHA-224, SHA-256, SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP192r1</td>
<td>96</td>
<td>SHA-224, SHA-256, SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP224r1</td>
<td>112</td>
<td>SHA-224, SHA-256, SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP256r1</td>
<td>128</td>
<td>SHA-256, SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP320r1</td>
<td>160</td>
<td>SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP384r1</td>
<td>192</td>
<td>SHA-384, SHA-512</td>
</tr>
<tr>
<td>brainpoolP512r1</td>
<td>256</td>
<td>SHA-512</td>
</tr>
</tbody>
</table>

Table 1

Security properties of the elliptic curve domain parameters specified in this RFC are discussed in Section 2.1. Further security discussions specific to elliptic curve cryptography can be found in [ANSII] and [SEC1].

6. Intellectual Property Rights

The authors have no knowledge about any intellectual property rights that cover the usage of the domain parameters defined herein. However, readers should be aware that implementations based on these domain parameters may require use of inventions covered by patent rights.

7. References

7.1. Normative References

7.2. Informative References


Appendix A. Pseudo-Random Generation of Parameters

In this appendix, the methods used for pseudo-random generation of the elliptic curve domain parameters are described. A comprehensive description is given in [EBP].

Throughout this section the following conventions are used:

The conversion between integers \( x \) in the range \( 0 \leq x \leq 2^L - 1 \) and bit strings of length \( L \) is given by \( x \leftrightarrow \{x_1, \ldots, x_L\} \) and the binary expansion

\[
x = x_1 \times 2^{(L-1)} + x_2 \times 2^{(L-2)} + \ldots + x_{(L-1)} \times 2 + x_L,\text{i.e., the first bit of the bit string corresponds to the most significant bit of the corresponding integer and the last bit to the least significant bit.}
\]

For a real number \( x \), let \( \text{floor}(x) \) denote the highest integer less than or equal to \( x \).

For updating the seed \( s \) of 160-bit length we use the following function \( \text{update_seed}(s) \):

1. Convert \( s \) to an integer \( z \).
2. Convert \( (z+1) \mod 2^{160} \) to a bit string \( t \) and output \( t \).

A.1. Generation of Prime Numbers

This section describes the systematic selection of the base fields \( \text{GF}(p) \) proposed in this specification. The prime generation method is similar to the method given in FIPS 186-2 [FIPS], Appendix 6.4, and ANSI X9.62 [ANSI1], A.3.2. It is a modification of the method "incremental search" given in Section 8.2.2 of [ISO3].

For computing an integer \( x \) in the range \( 0 \leq x \leq 2^L - 1 \) from a seed \( s \) of 160-bit length, we use the following algorithm \( \text{find_integer}(s) \):

1. Set \( v = \text{floor}((L-1)/160) \) and \( w = L - 160 \times v \).
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. Convert \( s \) to an integer \( z \).
5. For \( i \) from 1 to \( v \) do:
A. Set \( z_i = (z+i) \mod 2^{160} \).

B. Convert \( z_i \) to a bit string \( s_i \).

C. Set \( h_i = \text{SHA-1}(s_i) \).

6. Let \( h \) be the string obtained by the concatenation of \( h_0, \ldots, h_v \) from left to right.

7. Convert \( h \) to an integer \( x \) and output \( x \).

The following procedure is used to generate an \( L \) bit prime \( p \) from a 160-bit seed \( s \).

1. Set \( c = \text{find_integer}(s) \).

2. Let \( p \) be the smallest prime \( p \geq c \) with \( p = 3 \mod 4 \).

3. If \( 2^{L-1} \leq p \leq 2^L - 1 \) output \( p \) and stop.

4. Set \( s = \text{update_seed}(s) \) and go to Step 1.

For the generation of the primes \( p \) used as base fields \( \text{GF}(p) \) for the curves defined in this specification (and the corresponding twisted curves), the following values (in hexadecimal representation) have been used as initial seed \( s \):

- Seed\_p\_160 for \text{brainpoolP160r1}:
  3243F6A8885A308D313198A2E03707344A409382

- Seed\_p\_192 for \text{brainpoolP192r1}:
  2299F31D0082EFA98EC4E6C89452821E638D0137

- Seed\_p\_224 for \text{brainpoolP224r1}:
  7BE5466CF34E90C6CC0AC29B7C97C50DD3F84D5B

- Seed\_p\_256 for \text{brainpoolP256r1}:
  5B54709179216D5D98979FB1BD1310BA698DFB5A

- Seed\_p\_320 for \text{brainpoolP320r1}:
  C2FFD72DBD01ADFB7B8E1AFED6A267E96BA7C904

- Seed\_p\_384 for \text{brainpoolP384r1}:
  5F12C7F9924A19947B3916CF70801F2E2858EFC1

- Seed\_p\_512 for \text{brainpoolP512r1}:
  6636920D871574E69A458FEA3F4933D7E0D95748
These seeds have been obtained as the first 7 substrings of 160-bit length each of \( Q = \pi \times 2^{1120} \), where \( \pi \) is the constant 3.14159..., also known as Ludolph’s number, i.e.,

\[
Q = \text{Seed}_p_{160}||\text{Seed}_p_{192}||...||\text{Seed}_p_{512}||\text{Remainder},
\]

where \( || \) denotes concatenation.

Using these seeds and the above algorithm the following primes are obtained:

\[
\begin{align*}
 p_{160} & = 1332297598440044874827085558802491743757193798159 \\
 p_{192} & = 4781668983906166242955001894344923773259119655253013193367 \\
 p_{224} & = 227216229324535278755253799591092807334073214594492304435472941311 \\
 p_{256} & = 76884956397045344220809746629001649093037950200943055203735601445031516197751 \\
 p_{320} & = 17635933222391663541619098424460195208895127727195151927729604152886408688021498180955014999035278 \\
 p_{384} & = 2165927077011931617306923684232604979796116387017648600081618503821089934025961822236561982844534088440708417973331 \\
 p_{512} & = 8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947
\end{align*}
\]

A.2. Generation of Pseudo-Random Curves

The generation procedure is similar to the procedure given in FIPS PUB 186-2 [FIPS], Appendix 6.4, and ANSI X9.62 [ANSII], A.3.2.

For computing an integer \( x \) in the range \( 0 \leq x \leq 2^{(L-1)} - 1 \) from a seed \( s \) of 160-bit length, we use the algorithm find_integer_2(s), which slightly differs from the method used for the generation of the primes.

1. Set \( v = \text{floor}((L-1)/160) \) and \( w = L - 160 \times v - 1 \).
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. Convert \( s \) to an integer \( z \).
5. For i from 1 to v do:
   A. Set \( z_i = (z+i) \mod 2^{160} \).
   B. Convert \( z_i \) to a bit string \( s_i \).
   C. Set \( h_i = \text{SHA-1}(s_i) \).

6. Let \( h \) be the string obtained by the concatenation of \( h_0, \ldots, h_v \) from left to right.

7. Convert \( h \) to an integer \( x \) and output \( x \).

The following procedure is used to generate the parameters \( A \) and \( B \) of a suitable elliptic curve over \( \text{GF}(p) \) and a base point \( G \) from a prime \( p \) of bit length \( L \) and a 160-bit seed \( s \).

1. Set \( h = \text{find_integer}_2(s) \).

2. Convert \( h \) to an integer \( A \).

3. If \(-3 = A \cdot Z^4 \mod p\) is not solvable, then set \( s = \text{update_seed}(s) \) and go to Step 1.

4. Compute one solution \( Z \) of \(-3 = A \cdot Z^4 \mod p\).

5. Set \( s = \text{update_seed}(s) \).

6. Set \( B = \text{find_integer}_2(s) \).

7. If \( B \) is a square \( \mod p \), then set \( s = \text{update_seed}(s) \) and go to Step 6.

8. If \( 4 \cdot A^3 + 27 \cdot B^2 = 0 \mod p \), then set \( s = \text{update_seed}(s) \) and go to Step 1.

9. Check that the elliptic curve \( E \) over \( \text{GF}(p) \) given by \( y^2 = x^3 + A \cdot x + B \) fulfills all security and functional requirements given in Section 3. If not, then set \( s = \text{update_seed}(s) \) and go to Step 1.

10. Set \( s = \text{update_seed}(s) \).

11. Set \( k = \text{find_integer}_2(s) \).

12. Determine the points \( Q \) and \(-Q\) having the smallest \( x \)-coordinate in \( E(\text{GF}(p)) \). Randomly select one of them as point \( P \).
13. Compute the base point \( G = k \cdot P \).

14. Output \( A, B, \) and \( G \).

Note: Of course \( P \) could also be used as a base point. However, the small x-coordinate of \( P \) could possibly render the curve vulnerable to side-channel attacks.

For the generation of curve parameters \( A \) and \( B \), and the base points \( G \) defined in this specification, the following values (in hexadecimal representation) have been used as initial seed \( s \):

- **Seed_ab_160** for brainpoolP160r1: 2B7E151628AED2A6ABF7158809CF4F3C762E7160
- **Seed_ab_192** for brainpoolP192r1: F38B4DA56A784D9045190CFEF324E7738926CFBE
- **Seed_ab_224** for brainpoolP224r1: 5F4BF8D8D8C31D763DA06C80ABB1185EB4F7C7B5
- **Seed_ab_256** for brainpoolP256r1: 757F5958490CFD47D7C19BB42158D9554F7B46BC
- **Seed_ab_320** for brainpoolP320r1: ED55C4D79FD5F24D6613C31C3839A2DDF8A9A276
- **Seed_ab_384** for brainpoolP384r1: BCF8FA1C877C56284DAB79CD4C2B3293D20E9E5E
- **Seed_ab_512** for brainpoolP384r1: AF02AC60ACC93ED874422A52E8238FEE5AB6AD

These seeds have been obtained as the first 7 substrings of 160-bit length each of \( R = \text{floor}(e \times 2^{1120}) \), where \( e \) denotes the constant 2.71828..., also known as Euler’s number, i.e.,

\[
R = \text{Seed_ab}_160 || \text{Seed_ab}_192 || \ldots || \text{Seed_ab}_512 || \text{Remainder},
\]

where \( || \) denotes concatenation.
Authors’ Addresses

Manfred Lochter
Bundesamt fuer Sicherheit in der Informationstechnik (BSI)
Postfach 200363
53133 Bonn
Germany

Phone: +49 228 9582 5643
EMail: manfred.lochter@bsi.bund.de

Johannes Merkle
secunet Security Networks
Mergenthaler Allee 77
65760 Eschborn
Germany

Phone: +49 201 5454 2021
EMail: johannes.merkle@secunet.com