Research on OpenSSL Elliptic Curves for Compliance with the Russian National Digital Signature Standard

Stanislav S. Malakhov[1]

[1]National Research University Higher School of Economics

January 31, 2020

Abstract

The survey deals with elliptic curves which are implemented in the OpenSSL 1.1.1d software library. The objective of this work is to highlight the elliptic curves which comply with the Russian national digital signature standard, namely the GOST R 34.10–2012. For this reason the paper focuses on the OpenSSL elliptic curves over a finite field of a prime order and provides the results of testing those curves for compliance with the GOST R 34.10–2012 requirements. Two cases are observed. The first case covers a complete set of restrictions imposed on elliptic curves parameters, whereas the second one differs in that a restriction on a bit length of a number of points on the curve is omitted. For both cases the paper presents tables which list the curves tested along with corresponding match marks. In order to conduct tests the Wolfram Mathematica computing system was employed, and the Wolfram language source code is given in the appendices. Note, that the paper does not address to a rationale of the requirements of the standard nor does it focus on the parameters generation issues.

Keywords— elliptic curve, Russian digital signature standard, GOST, GOST R 34.10–2012, OpenSSL

Introduction

Currently, the cryptographic algorithms, which security strength is based upon the elliptic curve discrete logarithm problem, become more widespread. The two instances are the ECDSA scheme [1, 6] and the Russian digital signature standard GOST R 34.10–2012 [3]. An increasing concern of an elliptic curve cryptography can be explained by the fact that the same security strength is achieved by utilizing private keys of a smaller bit length, comparing to the algorithms based upon the finite field discrete logarithm problem and the integer factorization problem [7]. At present, only the exponential-time algorithms that solve the elliptic curve discrete logarithm problem in general case are known. Nevertheless, the security strength of cryptographic algorithms, that exploit elliptic curves, essentially depends on the chosen curve parameters. Several types of curves are known to provide extra potential for analysis, resulting in complexity reduction of solving the discrete logarithm problem [1].

One of the cryptographic libraries that implement the elliptic curve cryptography is OpenSSL, an open-source software library which is embedded in the set of Russian software. In particular, OpenSSL implements several elliptic curves defined in the standards [1, 2, 5, 6], and those curves can be utilized in digital signature schemes. For this reason the research on the OpenSSL elliptic curves for compliance with the GOST R 34.10–2012 requirements appears to be appropriate.

One may note that the Russian digital signature standard deals with elliptic curves over finite fields of prime orders only. Furthermore, there exist restrictions on a bit length of the number of points on a curve, seemingly, as a result of the fact that the GOST specifies the underlying hash functions with 256 bit or 512 bit hashes. Therefore, this work concentrates on the curves over prime fields, and in order to expand the scope of present work, the survey considers two cases, including one which takes a bit length constraint into account, and the other one which does not. Note, that current paper extends the results of [9].

This paper comprises two sections and two appendices. Section 1 presents the GOST R 34.10–2012 requirements, introduces an algorithm to test elliptic curves for compliance with those requirements and lists the results of the OpenSSL curves test. Section 2 does the same for the so called weak GOST R 34.10–2012 requirements. Appendix A contains a source code of routines that implement the test algorithms, while Appendix 1 lists the parameters of the elliptic curves tested.

1 GOST R 34.10–2012 requirements

We shall begin with defining the GOST R 34.10–2012 requirements which include a constraint on a bit length of the number of points.

Conforming to [3], an elliptic curve $\mathcal{E}$ over a finite field $\mathbb{F}_p$ of a prime order $p > 3$ is a set of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ of an affine plane together with the point $\mathcal{O}$ at infinity such that the following equation holds:

$$y^2 = x^3 + ax + b \pmod{p}, \tag{1}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

The Russian digital signature standard GOST R 34.10–2012 defines the following parameters of the digital signature scheme:

- an integer m, a number of points on a curve $\mathcal{E}$,
- a prime number q, an order of the chosen group of points on the curve $\mathcal{E}$ and
- a point $P = (x_p, y_p) \in \mathcal{E}$, a primitive element of a group of points on $\mathcal{E}$ such that $\mathrm{ord}(P) = q$.

The GOST imposes constraints on the aforementioned parameters:

$$\begin{gathered} q \text{ is prime}, \\ q \in \left(2^{254}; 2^{256}\right) \cup \left(2^{508}; 2^{512}\right), \\ p^t \not\equiv 1 \pmod{q}, t \in \{1, \ldots, 31\}, \text{if } q \in \left(2^{254}; 2^{256}\right), \\ p^t \not\equiv 1 \pmod{q}, t \in \{1, \ldots, 131\}, \text{if } q \in \left(2^{508}; 2^{512}\right), \\ m \neq p, \\ J(\mathcal{E}) = 1728 \cdot 4a^3 \left(4a^3 + 27b^2\right)^{-1}_{(\mathrm{mod}\ p)} \notin \{0, 1728\}. \end{gathered} \tag{2}$$

Definition 1. *We shall refer to the constraints (2) as the GOST R 34.10–2012 requirements, and if for the parameters of some elliptic curve the expressions (2) hold, then we shall say that such an elliptic curve satisfies the GOST R 34.10–2012 requirements.*

1.1 Algorithm to test for compliance with the GOST R 34.10–2012 requirements

To test whether selected elliptic curve satisfies the requirements of the GOST, a Wolfram Language program was developed; the corresponding source code is given in the appendix A. The program implements the following algorithm.

Step 1. Unless p is a prime number, proceed to Step 10.

Step 2. Unless q is a prime number, proceed to Step 10.

Step 3. Unless $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, proceed to Step 10.

STEP 4. Unless a point P lies on the elliptic curve $\mathcal{E}$, proceed to STEP 10.

STEP 5. Unless a point $qP = \mathcal{O}$, proceed to STEP 10.

STEP 6. Unless $q \in \left(2^{254}; 2^{256}\right) \cup \left(2^{508}; 2^{512}\right)$, proceed to STEP 10.

STEP 7. Unless

$$p^t \not\equiv 1 \pmod{q}, \quad \begin{cases} t \in \{1, \dots, 31\}, & \text{if } q \in \left(2^{254}; 2^{256}\right), \\ t \in \{1, \dots, 131\}, & \text{if } q \in \left(2^{508}; 2^{512}\right), \end{cases}$$

proceed to STEP 10.

STEP 8. Unless $J(\mathcal{E}) \notin \{0, 1728\}$, proceed to STEP 10.

STEP 9. Conclude that the elliptic curve $\mathcal{E}$ satisfies the GOST R 34.10–2012 requirements and terminate the algorithm execution.

STEP 10. Conclude that the elliptic curve $\mathcal{E}$ does not satisfy the GOST R 34.10–2012 requirements and terminate the algorithm execution.

1.2 Choice of elliptic curves for testing

It can be shown that only elliptic curves over $\mathbb{F}_p$ with a 254 to 257 or 508 to 513 bit length of order p might satisfy requirements of the Russian standard.

Indeed, the result of Hasse's theorem on elliptic curves [4, Theorem 3.7] leads to an estimate of a number of points on a curve $\mathcal{E}$:

$$p + 1 - 2\sqrt{p} \le |\mathcal{E}| = m \le p + 1 + 2\sqrt{p}.$$

Hence, the following holds:

$$\begin{cases} p + 1 - 2\sqrt{p} \le m \le p + 1 + 2\sqrt{p}, \\ \left[\begin{aligned} 2^{254} &< m < 2^{256}, \\ 2^{508} &< m < 2^{512}; \end{aligned}\right. \end{cases} \Longrightarrow \begin{cases} p + 1 + 2\sqrt{p} > 2^{254}, \\ p + 1 - 2\sqrt{p} < 2^{256}, \\ 2^{254} < m < 2^{256}; \end{cases} \text{OR} \begin{cases} p + 1 + 2\sqrt{p} > 2^{508}, \\ p + 1 - 2\sqrt{p} < 2^{512}, \\ 2^{508} < m < 2^{512}; \end{cases} \Longleftrightarrow$$

$$\left[\begin{aligned} &\begin{cases} 2^{254} - 2^{128} + 1 < p < 2^{256} + 2^{129} + 1, \\ 2^{254} < m < 2^{256}; \end{cases} \\ &\begin{cases} 2^{508} - 2^{255} + 1 < p < 2^{512} + 2^{257} + 1, \\ 2^{508} < m < 2^{512}; \end{cases} \end{aligned}\right. \tag{3}$$

Thereby, only a curve over a prime field $\mathbb{F}_p$ with p, satisfying inequalities (3), might meet the requirements of the GOST. For OpenSSL these are the curves over $\mathbb{F}_p$, where p has a 256 or 512 bit length. Once again shall we note the GOST does not permit exploitation of elliptic curves over non-prime fields.

1.3 Test results for compliance with the GOST R 34.10–2012 requirements

Table 1 presents a list of the OpenSSL elliptic curves tested together with the match marks that indicate whether a corresponding elliptic curve satisfies the GOST R 34.10–2012 requirements.

Elliptic curve identifier	Document	Match mark	Comments
secp256k1	[2, Section 2.7.1]	Does not satisfy	$J(\mathcal{E}) = 0$
prime256v1	[1, Section J.5.3]; [2, Section 2.7.2]; [6, Section D.1.2.3]	Satisfies	
brainpoolP256r1	[5, Section 3.4]	Satisfies	
brainpoolP256t1	[5, Section 3.4]	Satisfies	
brainpoolP512r1	[5, Section 3.7]	Satisfies	
brainpoolP512t1	[5, Section 3.7]	Satisfies	

Table 1: List of the OpenSSL elliptic curves tested for compliance with the GOST requirements and the corresponding match marks

2 Weak GOST R 34.10–2012 requirements

By analogy with Section 1 we shall introduce the weak GOST R 34.10–2012 requirements. Given the equation (1) and the same notation for parameters of the digital signature scheme, one may impose constraints on the parameters as follows:

$$\begin{gathered} q \text{ is prime}, \\ p^t \not\equiv 1 \pmod q, t \in \{1, \dots, 31\}, \text{if } q < 2^{256}, \\ p^t \not\equiv 1 \pmod q, t \in \{1, \dots, 131\}, \text{if } q \geq 2^{256}, \\ m \neq p, \\ J(\mathcal{E}) = 1728 \cdot 4a^3 \left(4a^3 + 27b^2\right)^{-1}_{\pmod p} \notin \{0, 1728\}. \end{gathered} \tag{4}$$

Definition 2. *We shall refer to the constraints (4) as the weak GOST R 34.10–2012 requirements, and if for the parameters of some elliptic curve the expressions (4) hold, then we shall say that such an elliptic curve satisfies the weak GOST R 34.10–2012 requirements.*

2.1 Algorithm to test for compliance with the weak GOST R 34.10–2012 requirements

To test whether selected elliptic curve satisfies the weak GOST R 34.10–2012 requirements, one may make certain changes to the algorithm in Section 1.1 in the following manner.

Step 1. Unless p is a prime number, proceed to Step 9.

Step 2. Unless q is a prime number, proceed to Step 9.

Step 3. Unless $4a^3 + 27b^2 \not\equiv 0 \pmod p$, proceed to Step 9.

Step 4. Unless a point P lies on the elliptic curve $\mathcal{E}$, proceed to Step 9.

Step 5. Unless a point $qP = \mathcal{O}$, proceed to Step 9.

Step 6. Unless

$$p^t \not\equiv 1 \pmod q, \quad \begin{cases} t \in \{1, \dots, 31\}, & \text{if } q < 2^{256} \\ t \in \{1, \dots, 131\}, & \text{if } q \geqslant 2^{256}, \end{cases}$$

proceed to Step 9.

Step 7. Unless $J(\mathcal{E}) \notin \{0, 1728\}$, proceed to Step 9.

Step 8. Conclude that the elliptic curve $\mathcal{E}$ satisfies the weak GOST R 34.10–2012 requirements and terminate the algorithm execution.

STEP 9. Conclude that the elliptic curve $\mathcal{E}$ does not satisfy the weak GOST R 34.10–2012 requirements and terminate the algorithm execution.

A Wolfram language source code of a program that implements the aforementioned algorithm is given in the appendix A.

2.2 Test results for compliance with the weak GOST R 34.10–2012 requirements

Table 2 presents a list of the OpenSSL elliptic curves tested together with the match marks that indicate whether a corresponding elliptic curve satisfies the weak GOST R 34.10–2012 requirements.

Elliptic curve identifier	Document	Match mark	Comments
secp112r1/ wap-wsg-idm-ecid-wtls6	[2, Section 2.2.1]; [8, Table 8]	Satisfies	
secp112r2	[2, Section 2.2.2]	Satisfies	
secp128r1	[2, Section 2.3.1]	Satisfies	
secp128r2	[2, Section 2.3.2]	Satisfies	
secp160k1	[2, Section 2.4.1]	Does not satisfy	$J(\mathcal{E}) = 0$
secp160r1/ wap-wsg-idm-ecid-wtls7	[2, Section 2.4.2]; [8, Table 8]	Satisfies	
secp160r2	[2, Section 2.4.3]	Satisfies	
secp192k1	[2, Section 2.5.1]	Does not satisfy	$J(\mathcal{E}) = 0$
secp224k1	[2, Section 2.6.1]	Does not satisfy	$J(\mathcal{E}) = 0$
secp224r1/ wap-wsg-idm-ecid-wtls12	[2, Section 2.6.2]; [6, Section D.2.2]; [8, Table 8]	Satisfies	
secp384r1	[2, Section 2.8.1]; [6, Section D.2.4]	Satisfies	
prime192v1	[1, Section J.5.1]; [2, Section 2.5.1]; [6, Section D.2.1]	Satisfies	
prime192v2	[1, Section J.5.1]	Satisfies	
prime192v3	[1, Section J.5.1]	Satisfies	
prime239v1	[1, Section J.5.2]	Satisfies	
prime239v2	[1, Section J.5.2]	Satisfies	
prime239v3	[1, Section J.5.2]	Satisfies	
wap-wsg-idm-ecid-wtls8	[8, Table 8]	Satisfies	
wap-wsg-idm-ecid-wtls9	[8, Table 8]	Satisfies	
brainpoolP160r1	[5, Section 3.1]	Satisfies	
brainpoolP160t1	[5, Section 3.1]	Satisfies	
brainpoolP192r1	[5, Section 3.2]	Satisfies	
brainpoolP192t1	[5, Section 3.2]	Satisfies	
brainpoolP224r1	[5, Section 3.3]	Satisfies	
brainpoolP224t1	[5, Section 3.3]	Satisfies	
brainpoolP320r1	[5, Section 3.6]	Satisfies	
brainpoolP320t1	[5, Section 3.6]	Satisfies	
brainpoolP384r1	[5, Section 3.7]	Satisfies	
brainpoolP384t1	[5, Section 3.7]	Satisfies	

Table 2: List of the OpenSSL elliptic curves tested for compliance with the weak GOST requiremets and the corresponding match marks

3 Conclusion

The survey comprises the research on the OpenSSL 1.1.1d elliptic curves for compliance with the GOST R 34.10–2012 requirements. The research contains the experimental results obtained by executing the specifically developed Wolfram language program. These results demonstrate that only the curves, which have a coefficient $a = 0$, do not comply with the requirements, when the additional formal restriction on a bit length of the number of points on a curve is omitted.

References

[1] American National Standards Institute (ANSI). *Working Draft ANSI X9.62-1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm*. Sept. 1998.

[2] D. R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Certicom Corp. Accessed at January 31, 2020, `https://www.secg.org/sec2-v2.pdf`. Jan. 2010.

[3] *GOST R 34.10–2012: Information technology. Cryptographic data security. Signature and verification processes of [electronic] digital signature*. (in Russian). Aug. 2012.

[4] D. Hankerson, A. Menezes, and S. Vanstone. *Guide To Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., 2004.

[5] M. Lochter and J. Merkle. *RFC 5639 (Request for Comments): Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. The Internet Society. Accessed at January 31, 2020, `https://tools.ietf.org/html/rfc5639`. Mar. 2010.

[6] National Institute of Standards and Technology (NIST). *FIPS PUB 186-4 (Federal information processing standards publication): Digital Signature Standard (DSS)*. Accessed at January 31, 2020, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`. July 2013.

[7] National Institute of Standards and Technology (NIST). *NIST Special Publication 800-57 Part 1 Revision 4: Recommendation for Key Management*. Accessed at January 31, 2020, `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf`. Jan. 2016.

[8] *Wireless Transport Layer Security. Wireless Application Protocol*. Wireless Application Protocol Forum Ltd. Accessed at January 31, 2020, `http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf`. Apr. 2001.

[9] С. С. Малахов. "Исследование Эллиптических Кривых из Зарубежных Стандартов на Соответствие Требованиям ГОСТ Р 34.10-2012 [Research on Elliptic Curves from the Foreign Standards for Compliance with the GOST R 34.10–2012]". In: *Межвузовская научно-техническая конференция студентов, аспирантов и молодых специалистов им. ЕВ Арменского*. (in Russian). 2019, pp. 208–209.

Appendix A Source code of routines

```
(* This function computes affine coordintes of (2 * point), utilizing
    coefficient a and a prime p. *)
Clear[AffinePointDoublingGOST];
AffinePointDoublingGOST[point_,a_,p_]:=Module[{L,x1,xr,y1,yr},
  If[Abs@point==Infinity,Return[Infinity]];
  x1=point[[1]]; y1=point[[2]];
```

```
  If[Mod[ y1,p]==0,Return[Infinity],L=Mod[(3x1^2+a)*ModularInverse[2
     y1,p],p];
    xr=Mod[L^2-2x1,p];
    yr=Mod[L(x1-xr)-y1,p]
  ];
  Return[{xr,yr}];
]

(* This function computes affine coordintes of (point1 + point2),
   utilizing coefficient a and a prime p. *)
Clear[AffinePointAdditionGOST];
AffinePointAdditionGOST[point1_,point2_,a_,p_]:=Module[{L,x1,x2,xr,y1
   ,y2,yr},
  If[Abs@point1==Infinity,Return[point2]];
  If[Abs@point2==Infinity,Return[point1]];
  x1=point1[[1]]; y1=point1[[2]];
  x2=point2[[1]];y2=point2[[2]];
  If[y1==Mod[-y2,p],Return[Infinity]];
  If[point1==point2,Return[AffinePointDoublingGOST[point1,a,p]],
    L=Mod[(y2-y1)*ModularInverse[x2-x1,p],p];
    xr=Mod[L^2-x1-x2,p];
    yr=Mod[L(x1-xr)-y1,p]
  ];
  Return[{xr,yr}];
]

(* This function computes affine coordintes of (k * point), utilizing
    coefficient a and a prime p. *)
Clear[AffinePointExponentiation];
AffinePointExponentiation[k_, point_, a_, p_] := Module[{K, Q},
  K = IntegerDigits[k, 2];
  Q = Infinity;
  For[i = 1, i <= Length@K, i++,
    Q = AffinePointDoublingGOST[Q, a, p];
    If[K[[i]] == 1, Q = AffinePointAdditionGOST[Q, point, a, p]]
  ];
  Return[Q];
]

(* This function tests if an elliptic curve over a finite field of a
   prime order satisfies the GOST 34.10-2012 requirements. Here p is
   the finite field order; q is an order of a cyclic subgroup of an
   elliptic curve group; n is a cofactor, that is n = m/q; a and b
   are coefficients of the elliptic curve equation. *)
Clear[CurveSatisfiesGOSTQ]
CurveSatisfiesGOSTQ[p_, q_, n_,a_, b_,x_,y_]:=Module[{satisfiesGOST,
   Invariant},
  satisfiesGOST=True;
  If[!PrimeQ[p],satisfiesGOST=False;Print["p is not prime"],
    If[!PrimeQ[q],satisfiesGOST=False;Print["q is not prime"],
      If[Mod[4*PowerMod[a,3,p]+27*PowerMod[b,2,p],p]==0,
        satisfiesGOST=False;
        Print["4a^3 + 27b^2 \[Congruent] 0(mod p)"],
        If[Mod[y^2,p]!=Mod[x^3+a x+b,p],satisfiesGOST=False;
```

```
          Print["Point (x;y) is not on the curve"],
          If[AffinePointExponentiation[q,{x,y},a,p]!=Infinity,
            satisfiesGOST=False;
            Print["q is not an order of the point (x;y)"],
            If[q<=2^254||2^256<=q<=2^508||q>=2^512,
              satisfiesGOST=False;
              Print["q has insufficient size"],
              If[2^254<q<2^256,
                Do[If[Mod[PowerMod[p,t,q],q]==1,
                  satisfiesGOST=False;
                  Print["Congruence does not hold for p and q: p^",t
                     ," \[Congruent] 1 (mod q)"];
                  Break[]],{t,1,31}],
                Do[If[Mod[PowerMod[p,t,q],q]==1,
                  satisfiesGOST=False;
                  Print["Congruence does not hold for p and q: p^",t
                     ," \[Congruent] 1 (mod q)"];
                  Break[]],{t,1,131}]
              ];
            ];
            If[satisfiesGOST==True,
              If[q*n==p,
                satisfiesGOST=False;
                Print["q equals p"],
                Invariant=Mod[4*PowerMod[12a,3,p]*ModularInverse[4*
                   PowerMod[a,3,p]+27*PowerMod[b,2,p],p],p];
                If[Invariant==0,
                  satisfiesGOST=False;
                  Print["Invariant equals 0"],
                  If[Invariant==1728,
                    satisfiesGOST=False;
                    Print["Invariant equals 1728"]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ];
  Return[satisfiesGOST];
]

(* This function tests if an elliptic curve over a finite field of a
   prime order satisfies the weak GOST 34.10-2012 requirements. Here
   p is the finite field characteristic; q is an order of a cyclic
   subgroup of an elliptic curve group; n is a cofactor, that is n =
   m/q; a and b are coefficients of the elliptic curve equation. *)
Clear[CurveSatisfiesGOSTQWeak];
CurveSatisfiesGOSTQWeak[p_, q_, n_,a_, b_,x_,y_]:=Module[{
   satisfiesGOST,Invariant},
  satisfiesGOST=True;
  If[!PrimeQ[p],satisfiesGOST=False;Print["p is not prime"],
```

```
    If[!PrimeQ[q],satisfiesGOST=False;Print["q is not prime"],
      If[Mod[4*PowerMod[a,3,p]+27*PowerMod[b,2,p],p]==0,
        satisfiesGOST=False;
        Print["4a^3 + 27b^2 \[Congruent] 0(mod p)"],
        If[Mod[y^2,p]!=Mod[x^3+a x+b,p],
          satisfiesGOST=False;
          Print["Point (x;y) is not on the curve"],
          If[AffinePointExponentiation[q,{x,y},a,p]!=Infinity,
            satisfiesGOST=False;
            Print["q is not an order of the point (x;y)"],
            Do[If[Mod[PowerMod[p,t,q],q]==1,
              satisfiesGOST=False;
              Print["Congruence does not hold for p and q: p^",t," \[
                 Congruent] 1 (mod q)"];
              Break[]],{t,1,Piecewise[{{31,q<2^256},{131,2^256<=q}}]}
            ];
            If[satisfiesGOST==True,
              If[q*n==p,
                satisfiesGOST=False;
                Print["q equals p"],
                Invariant=Mod[4*PowerMod[12a,3,p]*ModularInverse[4*
                   PowerMod[a,3,p]+27*PowerMod[b,2,p],p],p];
              If[Invariant==0,
                satisfiesGOST=False;
                Print["Invariant equals 0"],
                If[Invariant==1728,
                  satisfiesGOST=False;
                  Print["Invariant equals 1728"]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ];
  Return[satisfiesGOST];
]
```

Appendix B Parameters of the Elliptic curves tested

```
(* secp256k1 *)
p=Interpreter["HexInteger"]["
   FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
   "];
a=Interpreter["HexInteger
   "]["0000000000000000000000000000000000000000000000000000000000000000"];

b=Interpreter["HexInteger
   "]["0000000000000000000000000000000000000000000000000000000000000007"];

q=Interpreter["HexInteger"]["
```

```
   FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
   "];
x=Interpreter["HexInteger"]["79
   BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798"];
y=Interpreter["HexInteger"]["483
   ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8"];
CurveSatisfiesGOSTQ[p, q, 1, a, b,x,y]

(* prime256v1 *)
p = Interpreter["HexInteger"][
"FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B"];
q = Interpreter["HexInteger"][
"FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551"];
x = Interpreter["HexInteger"][
"6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296"];
y = Interpreter["HexInteger"][
"4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5"];
CurveSatisfiesGOSTQ[p, q, 1, a, b, x, y]

(* brainpoolP256r1 *)
p = Interpreter["HexInteger"][
"A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377"];
a = Interpreter["HexInteger"][
"7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9"];
b = Interpreter["HexInteger"][
"26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6"];
q = Interpreter["HexInteger"][
"A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7"];
x = Interpreter["HexInteger"][
"8BD2AEB9CB7E57CB2C4B482FFC81B7AFB9DE27E1E3BD23C23A4453BD9ACE3262"];
y = Interpreter["HexInteger"][
"547EF835C3DAC4FD97F8461A14611DC9C27745132DED8E545C1D54C72F046997"];
CurveSatisfiesGOSTQ[p, q, 1, a, b, x, y]

(* brainpoolP256t1 *)
p = Interpreter["HexInteger"][
"A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377"];
z = Interpreter["HexInteger"][
"3E2D4BD9597B58639AE7AA669CAB9837CF5CF20A2C852D10F655668DFC150EF0"];
a = Interpreter["HexInteger"][
"A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5374"];
b = Interpreter["HexInteger"][
"662C61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04"];
q = Interpreter["HexInteger"][
"A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7"];
x = Interpreter["HexInteger"][
"A3E8EB3CC1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F4"];
y = Interpreter["HexInteger"][
"2D996C823439C56D7F7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE"];
CurveSatisfiesGOSTQ[p, q, 1, a, b, x, y]
```

```
(* brainpoolP512r1 *)
p = Interpreter["HexInteger"][
"AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D\
4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3"];
a = Interpreter["HexInteger"][
"7830A3318B603B89E2327145AC234CC594CBDD8D3DF91610A83441CAEA9863BC2D\
ED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CA"];
b = Interpreter["HexInteger"][
"3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72B\
F2C7B9E7C1AC4D77FC94CADC083E67984050B75EBAE5DD2809BD638016F723"];
q = Interpreter["HexInteger"][
"AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA7033087055\
3E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069"];
x = Interpreter["HexInteger"][
"81AEE4BDD82ED9645A21322E9C4C6A9385ED9F70B5D916C1B43B62EEF4D0098EFF\
3B1F78E2D0D48D50D1687B93B97D5F7C6D5047406A5E688B352209BCB9F822"];
y = Interpreter["HexInteger"][
"7DDE385D566332ECC0EABFA9CF7822FDF209F70024A57B1AA000C55B881F8111B2\
DCDE494A5F485E5BCA4BD88A2763AED1CA2B2FA8F0540678CD1E0F3AD80892"];
CurveSatisfiesGOSTQ[p, q, 1, a, b, x, y]

(* brainpoolP512t1 *)
p = Interpreter["HexInteger"][
"AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D\
4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3"];
z = Interpreter["HexInteger"][
"12EE58E6764838B69782136F0F2D3BA06E27695716054092E60A80BEDB212B64E5\
85D90BCE13761F85C3F1D2A64E3BE8FEA2220F01EBA5EEB0F35DBD29D922AB"];
a = Interpreter["HexInteger"][
"AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D\
4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F0"];
b = Interpreter["HexInteger"][
"7CBBBCF9441CFAB76E1890E46884EAE321F70C0BCB4981527897504BEC3E36A62B\
CDFA2304976540F6450085F2DAE145C22553B465763689180EA2571867423E"];
q = Interpreter["HexInteger"][
"AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553\
E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069"]; x =
Interpreter["HexInteger"][
"640ECE5C12788717B9C1BA06CBC2A6FEBA85842458C56DDE9DB1758D39C0313D82B\
A51735CDB3EA499AA77A7D6943A64F7A3F25FE26F06B51BAA2696FA9035DA"];
y = Interpreter["HexInteger"][
"5B534BD595F5AF0FA2C892376C84ACE1BB4E3019B71634C01131159CAE03CEE9D9\
932184BEEF216BD71DF2DADF86A627306ECFF96DBB8BACE198B61E00F8B332"];
CurveSatisfiesGOSTQ[p, q, 1, a, b, x, y]

(* secp112r1/wap-wsg-idm-ecid-wtls6 *)
p = Interpreter["HexInteger"]["DB7C2ABF62E35E668076BEAD208B"];
a = Interpreter["HexInteger"]["DB7C2ABF62E35E668076BEAD2088"];
b = Interpreter["HexInteger"]["659EF8BA043916EEDE8911702B22"];
q = Interpreter["HexInteger"]["DB7C2ABF62E35E7628DFAC6561C5"];
x = Interpreter["HexInteger"]["09487239995A5EE76B55F9C2F098"];
y = Interpreter["HexInteger"]["A89CE5AF8724C0A23E0E0FF77500"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]
```

```
(* secp112r2 *)
p = Interpreter["HexInteger"]["DB7C2ABF62E35E668076BEAD208B"];
a = Interpreter["HexInteger"]["6127C24C05F38A0AAAF65C0EF02C"];
b = Interpreter["HexInteger"]["51DEF1815DB5ED74FCC34C85D709"];
q = Interpreter["HexInteger"]["36DF0AAFD8B8D7597CA10520D04B"];
x = Interpreter["HexInteger"]["4BA30AB5E892B4E1649DD0928643"];
y = Interpreter["HexInteger"]["ADCD46F5882E3747DEF36E956E97"];
CurveSatisfiesGOSTQWeak[p, q, 4, a, b, x, y]

(* secp128r1 *)
p = Interpreter["HexInteger"]["FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"]["FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFC"];
b = Interpreter["HexInteger"]["E87579C11079F43DD824993C2CEE5ED3"];
q = Interpreter["HexInteger"]["FFFFFFFE0000000075A30D1B9038A115"];
x = Interpreter["HexInteger"]["161FF7528B899B2D0C28607CA52C5B86"];
y = Interpreter["HexInteger"]["CF5AC8395BAFEB13C02DA292DDED7A83"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp128r2 *)
p = Interpreter["HexInteger"]["FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"]["D6031998D1B3BBFEBF59CC9BBFF9AEE1"];
b = Interpreter["HexInteger"]["5EEEFCA380D02919DC2C6558BB6D8A5D"];
q = Interpreter["HexInteger"]["3FFFFFFF7FFFFFFFBE0024720613B5A3"];
x = Interpreter["HexInteger"]["7B6AA5D85E572983E6FB32A7CDEBC140"];
y = Interpreter["HexInteger"]["27B6916A894D3AEE7106FE805FC34B44"];
CurveSatisfiesGOSTQWeak[p, q, 4, a, b, x, y]

(* secp160k1 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73"];
a = Interpreter["HexInteger"]["0"];
b = Interpreter["HexInteger"]["7"];
q = Interpreter["HexInteger"][
"0100000000000000000001B8FA16DFAB9ACA16B6B3"];
x = Interpreter["HexInteger"][
"3B4C382CE37AA192A4019E763036F4F5DD4D7EBB"];
y = Interpreter["HexInteger"][
"938CF935318FDCED6BC28286531733C3F03C4FEE"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp160r1/wap-wsg-idm-ecid-wtls7 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFC"];
b = Interpreter["HexInteger"][
"1C97BEFC54BD7A8B65ACF89F81D4D4ADC565FA45"];
q = Interpreter["HexInteger"][
"0100000000000000000001F4C8F927AED3CA752257"];
x = Interpreter["HexInteger"][
"4A96B5688EF573284664698968C38BB913CBFC82"];
y = Interpreter["HexInteger"][
"23A628553168947D59DCC912042351377AC5FB32"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]
```

```
(* secp160r2 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC70"];
b = Interpreter["HexInteger"][
"B4E134D3FB59EB8BAB57274904664D5AF50388BA"];
q = Interpreter["HexInteger"][
"0100000000000000000000351EE786A818F3A1A16B"];
x = Interpreter["HexInteger"][
"52DCB034293A117E1F4FF11B30F7199D3144CE6D"];
y = Interpreter["HexInteger"][
"FEAFFEF2E331F296E071FA0DF9982CFEA7D43F2E"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp192k1 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFEE37"];
a = Interpreter["HexInteger"]["0"];
b = Interpreter["HexInteger"]["3"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D"];
x = Interpreter["HexInteger"][
"DB4FF10EC057E9AE26B07D0280B7F4341DA5D1B1EAE06C7D"];
y = Interpreter["HexInteger"][
"9B2F2F6D9C5628A7844163D015BE86344082AA88D95E2F9D"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp224k1 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFE56D"];
a = Interpreter["HexInteger"]["0"];
b = Interpreter["HexInteger"]["5"];
q = Interpreter["HexInteger"][
"010000000000000000000000000001DCE8D2EC6184CAF0A971769FB1F7"];
x = Interpreter["HexInteger"][
"A1455B334DF099DF30FC28A169A467E9E47075A90F7E650EB6B7A45C"];
y = Interpreter["HexInteger"][
"7E089FED7FBA344282CAFBD6F7E319F7C0B0BD59E2CA4BDB556D61A5"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp224r1/wap-wsg-idm-ecid-wtls12 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000001"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFE"];
b = Interpreter["HexInteger"][
"B4050A850C04B3ABF54132565044B0B7D7BFD8BA270B39432355FFB4"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFF16A2E0B8F03E13DD29455C5C2A3D"];
x = Interpreter["HexInteger"][
"B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21"];
y = Interpreter["HexInteger"][
"BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34"];
```

```
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* secp384r1 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFF\
FFFFFF0000000000000000FFFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFF\
FFFFFF0000000000000000FFFFFFFC"];
b = Interpreter["HexInteger"][
"B3312FA7E23EE7E4988E056BE3F82D19181D9C6EFE8141120314088F5013875AC6\
56398D8A2ED19D2A85C8EDD3EC2AEF"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC7634D81F4372DDF58\
1A0DB248B0A77AECEC196ACCC52973"];
x = Interpreter["HexInteger"][
"AA87CA22BE8B05378EB1C71EF320AD746E1D3B628BA79B9859F741E082542A3855\
02F25DBF55296C3A545E3872760AB7"];
y = Interpreter["HexInteger"][
"3617de4a96262c6f5d9e98bf9292dc29f8f41dbd289a147ce9da3113b5f0b8c00a\
60b1ce1d7e819d7a431d7c90ea0e5f"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime192v1 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831"];
x = Interpreter["HexInteger"][
"188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012"];
y = Interpreter["HexInteger"][
"07192B95FFC8DA78631011ED6B24CDD573F977A11E794811"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime192v2 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"CC22D6DFB95C6B25E49C0D6364A4E5980C393AA21668D953"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFE5FB1A724DC80418648D8DD31"];
x = Interpreter["HexInteger"][
"EEA2BAE7E1497842F2DE7769CFE9C989C072AD696F48034A"];
y = Interpreter["HexInteger"][
"6574d11d69b6ec7a672bb82a083df2f2b0847de970b2de15"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime192v3 *)
```

```
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"22123DC2395A05CAA7423DAECCC94760A7D462256BD56916"];
q = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFF7A62D031C83F4294F640EC13"];
x = Interpreter["HexInteger"][
"7D29778100C65A1DA1783716588DCE2B8B4AEE8E228F1896"];
y = Interpreter["HexInteger"][
"38a90f22637337334b49dcb66a6dc8f9978aca7648a943b0"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime239v1 *)
p = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"6B016C3BDCF18941D0D654921475CA71A9DB2FB27D1D37796185C2942C0A"];
q = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF9E5E9A9F5D9071FBD1522688909D0B"];
x = Interpreter["HexInteger"][
"0FFA963CDCA8816CCC33B8642BEDF905C3D358573D3F27FBBD3B3CB9AAAF"];
y = Interpreter["HexInteger"][
"7debe8e4e90a5dae6e4054ca530ba04654b36818ce226b39fccb7b02f1ae"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime239v2 *)
p = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"617FAB6832576CBBFED50D99F0249C3FEE58B94BA0038C7AE84C8C832F2C"];
q = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF800000CFA7E8594377D414C03821BC582063"];
x = Interpreter["HexInteger"][
"38AF09D98727705120C921BB5E9E26296A3CDCF2F35757A0EAFD87B830E7"];
y = Interpreter["HexInteger"][
"5b0125e4dbea0ec7206da0fc01d9b081329fb555de6ef460237dff8be4ba"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* prime239v3 *)
p = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF"];
a = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFC"];
b = Interpreter["HexInteger"][
"255705FA2A306654B1F4CB03D6A750A30C250102D4988717D9BA15AB6D3E"];
q = Interpreter["HexInteger"][
"7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF975DEB41B3A6057C3C432146526551"];
x = Interpreter["HexInteger"][
```

```
"6768AE8E18BB92CFCF005C949AA2C6D94853D0E660BBF854B1C9505FE95A"];
y = Interpreter["HexInteger"][
"1607e6898f390c06bc1d552bad226f3b6fcfe48b6e818499af18e3ed6cf3"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* wap-wsg-idm-ecid-wtls8 *)
p = Interpreter["HexInteger"]["FFFFFFFFFFFFFFFFFFFFFFFFFDE7"];
a = Interpreter["HexInteger"]["0"];
b = Interpreter["HexInteger"]["3"];
q = Interpreter["HexInteger"]["100000000000001ECEA551AD837E9"];
x = Interpreter["HexInteger"]["1"];
y = Interpreter["HexInteger"]["2"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* wap-wsg-idm-ecid-wtls9 *)
p = Interpreter["HexInteger"][
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC808F"];
a = Interpreter["HexInteger"]["0"];
b = Interpreter["HexInteger"]["3"];
q = Interpreter["HexInteger"][
"100000000000000000001CDC98AE0E2DE574ABF33"];
x = Interpreter["HexInteger"]["1"];
y = Interpreter["HexInteger"]["2"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP160r1 *)
p = Interpreter["HexInteger"][
"E95E4A5F737059DC60DFC7AD95B3D8139515620F"];
a = Interpreter["HexInteger"][
"340E7BE2A280EB74E2BE61BADA745D97E8F7C300"];
b = Interpreter["HexInteger"]["\!\(\*
StyleBox[\"1\",\nFontWeight->\"Plain\"]\)\!\(\*
StyleBox[\"E589A8595423412134FAA2DBDEC95C8D8675E58\",\n\
FontWeight->\"Plain\"]\)"];
q = Interpreter["HexInteger"][
"E95E4A5F737059DC60DF5991D45029409E60FC09"];
x = Interpreter["HexInteger"][
"BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3"];
y = Interpreter["HexInteger"][
"1667CB477A1A8EC338F94741669C976316DA6321"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP160t1 *)
p = Interpreter["HexInteger"][
"E95E4A5F737059DC60DFC7AD95B3D8139515620F"];
a = Interpreter["HexInteger"][
"E95E4A5F737059DC60DFC7AD95B3D8139515620C"];
b = Interpreter["HexInteger"][
"7A556B6DAE535B7B51ED2C4D7DAA7A0B5C55F380"];
q = Interpreter["HexInteger"][
"E95E4A5F737059DC60DF5991D45029409E60FC09"];
x = Interpreter["HexInteger"][
"B199B13B9B34EFC1397E64BAEB05ACC265FF2378"];
y = Interpreter["HexInteger"][
```

```
"ADD6718B7C7C1961F0991B842443772152C9E0AD"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP192r1 *)
p = Interpreter["HexInteger"][
"C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297"];
a = Interpreter["HexInteger"][
"6A91174076B1E0E19C39C031FE8685C1CAE040E5C69A28EF"];
b = Interpreter["HexInteger"][
"469A28EF7C28CCA3DC721D044F4496BCCA7EF4146FBF25C9"];
q = Interpreter["HexInteger"][
"C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1"];
x = Interpreter["HexInteger"][
"C0A0647EAAB6A48753B033C56CB0F0900A2F5C4853375FD6"];
y = Interpreter["HexInteger"][
"14B690866ABD5BB88B5F4828C1490002E6773FA2FA299B8F"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP192t1 *)
p = Interpreter["HexInteger"][
"C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297"];
a = Interpreter["HexInteger"][
"C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86294"];
b = Interpreter["HexInteger"][
"13D56FFAEC78681E68F9DEB43B35BEC2FB68542E27897B79"];
q = Interpreter["HexInteger"][
"C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1"];
x = Interpreter["HexInteger"][
"3AE9E58C82F63C30282E1FE7BBF43FA72C446AF6F4618129"];
y = Interpreter["HexInteger"][
"097E2C5667C2223A902AB5CA449D0084B7E5B3DE7CCC01C9"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP224r1 *)
p = Interpreter["HexInteger"][
"D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF"];
a = Interpreter["HexInteger"][
"68A5E62CA9CE6C1C299803A6C1530B514E182AD8B0042A59CAD29F43"];
b = Interpreter["HexInteger"][
"2580F63CCFE44138870713B1A92369E33E2135D266DBB372386C400B"];
q = Interpreter["HexInteger"][
"D7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F"];
x = Interpreter["HexInteger"][
"0D9029AD2C7E5CF4340823B2A87DC68C9E4CE3174C1E6EFDEE12C07D"];
y = Interpreter["HexInteger"][
"58AA56F772C0726F24C6B89E4ECDAC24354B9E99CAA3F6D3761402CD"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP224t1 *)
p = Interpreter["HexInteger"][
"D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF"];
a = Interpreter["HexInteger"][
"D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FC"];
b = Interpreter["HexInteger"][
```

```
"4B337D934104CD7BEF271BF60CED1ED20DA14C08B3BB64F18A60888D"];
q = Interpreter["HexInteger"][
"D7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F"];
x = Interpreter["HexInteger"][
"6AB1E344CE25FF3896424E7FFE14762ECB49F8928AC0C76029B4D580"];
y = Interpreter["HexInteger"][
"0374E9F5143E568CD23F3F4D7C0D4B1E41C8CC0D1C6ABD5F1A46DB4C"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP320r1 *)
p = Interpreter["HexInteger"][
"D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FC\
D412B1F1B32E27"];
a = Interpreter["HexInteger"][
"3EE30B568FBAB0F883CCEBD46D3F3BB8A2A73513F5EB79DA66190EB085FFA9F492\
F375A97D860EB4"];
b = Interpreter["HexInteger"][
"520883949DFDBC42D3AD198640688A6FE13F41349554B49ACC31DCCD884539816F\
5EB4AC8FB1F1A6"];
q = Interpreter["HexInteger"][
"D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E986\
91555B44C59311"];
x = Interpreter["HexInteger"][
"43BD7E9AFB53D8B85289BCC48EE5BFE6F20137D10A087EB6E7871E2A10A599C710\
AF8D0D39E20611"];
y = Interpreter["HexInteger"][
"14FDD05545EC1CC8AB4093247F77275E0743FFED117182EAA9C77877AAAC6AC7D3\
5245D1692E8EE1"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP320t1 *)
p = Interpreter["HexInteger"][
"D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FC\
D412B1F1B32E27"];
a = Interpreter["HexInteger"][
"D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FC\
D412B1F1B32E24"];
b = Interpreter["HexInteger"][
"A7F561E038EB1ED560B3D147DB782013064C19F27ED27C6780AAF77FB8A547CEB5\
B4FEF422340353"];
q = Interpreter["HexInteger"][
"D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E986\
91555B44C59311"];
x = Interpreter["HexInteger"][
"925BE9FB01AFC6FB4D3E7D4990010F813408AB106C4F09CB7EE07868CC136FFF33\
57F624A21BED52"];
y = Interpreter["HexInteger"][
"63BA3A7A27483EBF6671DBEF7ABB30EBEE084E58A0B077AD42A5A0989D1EE71B1B\
9BC0455FB0D2C3"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP384r1 *)
p = Interpreter["HexInteger"][
"8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123AC\
```

```
D3A729901D1A71874700133107EC53"];
a = Interpreter["HexInteger"][
"7BC382C63D8C150C3C72080ACE05AFA0C2BEA28E4FB22787139165EFBA91F90F8A\
A5814A503AD4EB04A8C7DD22CE2826"];
b = Interpreter["HexInteger"][
"04A8C7DD22CE28268B39B55416F0447C2FB77DE107DCD2A62E880EA53EEB62D57C\
B4390295DBC9943AB78696FA504C11"];
q = Interpreter["HexInteger"][
"8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF\
3AB6AF6B7FC3103B883202E9046565"];
x = Interpreter["HexInteger"][
"1D1C64F068CF45FFA2A63A81B7C13F6B8847A3E77EF14FE3DB7FCAFE0CBD10E8E8\
26E03436D646AAEF87B2E247D4AF1E"];
y = Interpreter["HexInteger"][
"8ABE1D7520F9C2A45CB1EB8E95CFD55262B70B29FEEC5864E19C054FF99129280E\
4646217791811142820341263C5315"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]

(* brainpoolP384t1 *)
p = Interpreter["HexInteger"][
"8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123AC\
D3A729901D1A71874700133107EC53"];
a = Interpreter["HexInteger"][
"8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123AC\
D3A729901D1A71874700133107EC50"];
b = Interpreter["HexInteger"][
"7F519EADA7BDA81BD826DBA647910F8C4B9346ED8CCDC64E4B1ABD11756DCE1D20\
74AA263B88805CED70355A33B471EE"];
q = Interpreter["HexInteger"][
"8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF\
3AB6AF6B7FC3103B883202E9046565"];
x = Interpreter["HexInteger"][
"18DE98B02DB9A306F2AFCD7235F72A819B80AB12EBD653172476FECD462AABFFC4\
FF191B946A5F54D8D0AA2F418808CC"];
y = Interpreter["HexInteger"][
"25AB056962D30651A114AFD2755AD336747F93475B7A1FCA3B88F2B6A208CCFE46\
9408584DC2B2912675BF5B9E582928"];
CurveSatisfiesGOSTQWeak[p, q, 1, a, b, x, y]
```