A note on high-security general-purpose elliptic curves

Diego F. Aranha[1], Paulo S. L. M. Barreto[2*],
Geovandro C. C. F. Pereira[2], and Jefferson E. Ricardini[2]

[1] Computer Science Dept, University of Brasília.
E-mail: dfaranha@unb.br
[2] Escola Politécnica, University of São Paulo.
E-mails: {pbarreto,geovandro,jricardini}@larc.usp.br

Abstract. In this note we describe some general-purpose, high-efficiency elliptic curves tailored for security levels beyond 2^{128}. For completeness, we also include legacy-level curves at standard security levels. The choice of curves was made to facilitate state-of-the-art implementation techniques.

1 Introduction

General-purpose elliptic curves are necessary to attain high-efficiency implementations of the most common cryptographic protocols like asymmetric encryption and plain digital signatures (but setting aside less conventional application like identity-based encryption). The standard NIST curves [14], though fairly efficient overall, arguably no longer represent the state of the art in the area [4, 7].

More efficient general-purpose curves have been recently proposed to address this situation [3, 4, 8], but for the 2^{128} security level at most, which corresponds to the expected security level of the standard NIST curve P-256 (or its binary counterpart, B-283). This is the case of Curve25519 [3] and Curve1174 [4]. However, while there is reason to look for higher security curves [15], no similar curves seem to have been proposed in the literature for higher security levels[3], matching the presumed levels of the standard NIST curves P-384 and P-521, as well as their binary counterparts, B-409 and B-571.

In this note we address this need, adopting the same settings as Curve25519 and Curve1174, respectively. Specifically, we describe Elligator (1 and 2) curves with target security levels roughly matching the presumed levels of all standard NIST curves.

2 Curve choice

The curves Curve25519 and Curve1174 have been engineered to facilitate simple, efficient and secure implementation of general-purpose elliptic curve cryptosys-

[*] Supported by CNPq research productivity grant 306935/2012-0.

[3] However, since the initial version of this note was made public the SafeCurves site [5] has appeared with an extensive treatment of the subject.

tems, with impressive results [8]. On these grounds, it makes sense to look for similar curves at higher security levels. At the same time, one can take the opportunity to provide curves matching the expected security levels of other standard NIST curves, the most prominent case being P-224 (and its binary counterpart, B-233), corresponding approximately to the security level of 3DES [12].

Curve25519 [3] is an Elligator type 2 curve with the following properties (among others):

- It is a Montgomery curve [13] over a large prime field $\mathbb{F}_p$;
- The prime p has the form $p = 2^m - \delta$ where $0 < \delta < \lceil \lg(p) \rceil = m$;
- The prime p satisfies $p \equiv 5 \pmod 8$, hence square root computation in $\mathbb{F}_p$ can be done with the Atkin method [2];
- The value $\xi = 2$ is a quadratic non-residue in $\mathbb{F}_p$, and hence can used to define a non-trivial quadratic twist of an elliptic curve over $\mathbb{F}_p$;
- The curve equation is $E : y^2 = x^3 + Ax^2 + x$ and the twist equation is $E' : v^2 = u^3 + 2Au^2 + 4u$, where $|(A - 2)/4| > 0$ is as small as possible (so as to improve arithmetic performance), and $A^2 - 4$ is not a square (so that the curve has a unique point of order 2 and the Montgomery ladder yields a complete addition law).
- The curve order has the form $n = 8r$ where $r > 2^{m-3}$ is prime;
- The order of the non-trivial quadratic twist of the curve has the form $n' = 4r'$ where $r' > 2^{m-3}$ is prime, with $|r'| = |r| + 1$;

Curve1174 [4] is an Elligator type 1 curve with the following properties (among others):

- It is an Edwards curve [6, 10] over a large prime field $\mathbb{F}_p$;
- The prime p has the form $p = 2^m - \delta$ where $0 < \delta < \lceil \lg(p) \rceil = m$;
- The prime p satisfies $p \equiv 3 \pmod 4$, hence square root computation in $\mathbb{F}_p$ can be done with the Cippolla-Lehmer method [11];
- The curve equation is $E : x^2 + y^2 = 1 + dx^2y^2$ and the equation of a non-trivial quadratic twist of E is $E' : u^2 + v^2 = 1 + (1/d)u^2v^2$, where $|d| > 1$ is as small as possible (so as to improve arithmetic performance);
- The curve order has the form $n = 4r$ where $r > 2^{m-3}$ is prime;
- The order of the non-trivial quadratic twist of the curve has the form $n' = 4r'$ where $r' > 2^{m-3}$ is prime, with $|r'| = |r|$;

Besides, for the most part the bit sizes m were chosen to be the largest values strictly smaller than the commonplace sizes $\{224, 256, 384, 512\}$ such that a prime $p = 2^m - \delta$ as required above exists (except for previously existing Curve1174 which was defined with a slightly smaller bit size, namely, $m = 251$ rather than $m = 254$).

3 The curves

We now list curves for several security levels, up to the level roughly comparable to the presumed security level of the NIST curves P-521 and B-571. The primes have the general form $p = 2^m - \delta$ for δ as small as possible. While it would be desirable that $\delta < 32$ (see [4]), this is not always possible. Yet, insisting that $\delta < \lg p$ increases the likeliness that any attack advantage this setting might cause is negligible (exponentially small).

The naming convention we adopt follows the pattern "M-m" for the Montgomery curve uniquely defined by the conditions in Section 2 over an m-bit prime field, and "E-m" for the similarly unique Edwards curve. This is reminiscent of the NIST naming convention for curves. The literature [4, 7] suggests two other naming conventions, namely, "Curve$m\delta$" for the uniquely defined Montgomery curve over $\mathbb{F}_{2^m-\delta}$ (where δ is, in turn, uniquely defined by m), and "Curve$|d|$" for an Edwards curve with coefficient d (in this case, the underlying field is not apparent in the name). So, for instance, M-383 could also be called Curve383187, and E-521 could also be called Curve376014.

Table 1 contains curves in the Montgomery model, while Table 2 contains curves in the Edwards model. For completeness, we include Curve25519 and Curve1174 (which, by definition, satisfy the requirements in Section 2, but are labelled here as originally published). The prime group order is r. For the Edwards (Elligator 1) curves, the suggested base point of order r is given by its y coordinate while for Montgomery (Elligator 2) curves, is given by its x-coordinate. In each case, the given coordinate is as small as possible when viewed as a natural number. The 'ρ-sec' column indicates the security level as measured by the cost of a Pollard ρ attack.

Table 1. Montgomery curves

curve	p	A	ρ-sec	x	r
M-221	$2^{221} - 3$	117050	$2^{108.8}$	4	421249166674228746791672110734 68\ 216792689508198039630494433505 28\ 91
Curve25519	$2^{255} - 19$	486662	$2^{125.8}$	9	723700557733226221397318656304 29\ 942408571163593799076060019509 38\ 285454250989
M-383	$2^{383} - 187$	2065150	$2^{189.8}$	12	246262538727465495076744000625 89\ 758628174837044040904167469345 74\ 041288984234680883008327183083 61\ 5266784870011007447
M-511	$2^{511} - 187$	530438	$2^{253.8}$	5	837987995621412318723376562387 86\ 538296746036378702458610772259 02\ 326102518796074108048767793830 55\ 508762141059258497448934987052 50\ 877562616246093073794 2299

A proof-of-concept implementation of all these curves is available as part of the RELIC library [1]. Work on a production-quality implementation is ongoing. See [5] for more properties of these and other equally safe curves.

Table 2. Edwards curves

curve	p	d	ρ-sec	y	r
E-222	$2^{222} - 117$	160102	$2^{109.8}$	4	16849966666969149871666884429387\ 26735569737456760058294185521417\ 407
Curve1174	$2^{251} - 9$	-1174	$2^{124.3}$	4	90462569716653277674664832038037\ 42800923390352794954740234892617\ 73642975601
E-382	$2^{382} - 105$	-67254	$2^{189.8}$	13	24626253872746549507674400062589\ 75862817483704404090416745738034\ 55766305456464917126265932668324\ 4604346084081047321
E-521	$2^{521} - 1$	-376014	$2^{259.3}$	6	17161994150326524287454751997703\ 48304317358825035826352348615864\ 79638579584941367547587665166365\ 78496366936590652341426043192829\ 48702542317993421293670108523

4 Conclusion

We have described general-purpose high-efficiency curves roughly matching the expected security of the standard NIST curves P-384 and P-521, and as a bonus, also curves roughly matching the expected security of the standard NIST curve P-224. All curves provided herein follow the Elligator (1 and 2) strategy, which is arguably the state of the art for the design of cryptographically-oriented elliptic curves.

This is work in progress. Better curves may be suggested as they become available.

Acknowledgements

We are grateful to Samuel Neves for providing independent verification of the correctness of the curves included here (SAGE script available here: `http://eden.dei.uc.pt/~sneves/647.sage`), to Mike Scott for suggesting improvements for our own Magma scripts and providing independent verification of the curves herein in C/C++ (using the Miracl™ library), and to Mike Hamburg and Tanja Lange for pinpointing typos in a previous version of this note and suggesting many improvements.

References

1. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography.
2. O. Atkin. Square roots and cognate matters modulo $p = 8n + 5$. Number Theory mailing list, 1992. `http://listserv.nodak.edu/scripts/wa.exe?A2=ind9211&L=nmbrthry&O=T&P=562`.

3. Dan J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
4. Dan J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. IACR Cryptology ePrint Archive, report 2013/325, 2013.
5. Dan J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to`, 2013.
6. Dan J. Bernstein and Tanja Lange. Security dangers of the NIST curves. In K. Kurosawa, editor, *Advances in Cryptology – Asiacrypt 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007.
7. Dan J. Bernstein and Tanja Lange. Security dangers of the NIST curves. Invited talk, International State of the Art Cryptography Workshop, Athens, Greece, 2013.
8. Dan J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In Alejandro Hevia and Gregory Neven, editors, *Latincrypt 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2012.
9. Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).
10. Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.
11. D. H. Lehmer. Computer technology applied to the theory of numbers. In W. J. LeVeque, editor, *Studies in Number Theory*. Mathematical Association of America, 1969.
12. Stefan Lucks. Attacking triple encryption. In Serge Vaudenay, editor, *Fast Software Encryption – FSE 1998*, volume 1372 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 1998.
13. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.
14. National Institute of Standards and Technology – NIST. *Federal Information Processing Standard (FIPS 186-4) – Digital Signature Standard (DSS)*, July 2013.
15. National Security Agency – NSA. *Suite B Cryptography / Cryptographic Interoperability*, January 2009. `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`.
16. Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium*, San Francisco, CA, USA, 2011. USENIX Association.

A Legacy-level curves

Under certain circumstances, typically on legacy systems, where one needs a lower but still reasonable security level, or must adhere to layout constraints of existing applications, or simply cannot afford higher-security curves for lack of computational resources (as may be the case on certain very constrained platforms typical of the Internet of Things), curves matching the expected security level of (say) the NIST curves B-163 or P-192 may be useful.

Though the primary purpose of this note is to suggest some curves at high security levels, for completeness we list a few possible alternatives for those legacy-level curves on Tables 3 and 4. It is noteworthy to mention that one of the examples we provide here was inspired by the Telex system [16], where arithmetic is performed over the field $\mathbb{F}_p$ with $p = 2^{168} - 2^8 - 1$. Although this prime does not strictly satisfy the criterion $2^m - p < m$, it fails to do so only slightly, and it has the merit of enabling very efficient modular reduction, so we relax the criterion to $2^m - p < 2m$ in that case.

Table 3. Legacy-level Montgomery curves

curve	p	A	ρ-sec	x	r
M-159	$2^{159} - 91$	197782	$2^{77.8}$	3	913438523331814323877304111591l6\ 468190437625759
M-191	$2^{191} - 19$	-281742	$2^{93.8}$	11	39231885846166754773973683895833\ 3908942434975704595311597

Table 4. Legacy-level Edwards curves

curve	p	d	ρ-sec	y	r
E-157	$2^{157} - 133$	-42000	$2^{77.3}$	5	45671926166590716193865246478592\ 509883108923719
E-168	$2^{168} - 257$	-715	$2^{82.8}$	3	93536104789177786765035835538253\ 283032607241942227
E-191	$2^{190} - 33$	-15584	$2^{93.8}$	3	39231885846166754773973683896145\ 3290575684318090919900773

B Verifying the curves

The following Magma [9] script checks that the curves presented in this note do indeed satisfy the requirements in Section 2, except the condition that the coefficients A and d in the curve equations $E : y^2 = x^3 + Ax^2 + x$ and $E : x^2 + y^2 = 1 + dx^2y^2$ are as small as possible in absolute integer value. Extending the script so as to check this last condition is straightforward, but the processing time can be very long (several weeks for the highest security levels, if run sequentially). Independent verification has been kindly provided by Samuel Neves in Sage (script at `http://eden.dei.uc.pt/~sneves/647.sage`), and by Mike Scott in C/C++ using the Miracl™ library.

```
procedure MontyGenBasept(m, A)
    p := 2^m;
    repeat
        p := PreviousPrime(p : Proof := false);
    until p mod 8 eq 5;
    F := GF(p);
    A := F!A;
    E := EllipticCurve([0, A, 0, 1, 0]);
    n := #E;
    r := n div 8;
    assert n mod 8 eq 0 and IsProbablePrime(r);

    // find generator:
    x := F!0;
    repeat
        repeat
            x +:= F!1;
        until IsSquare(x^3 + A*x^2 + x);
        G := E![x, Sqrt(x^3 + A*x^2 + x)];
    until not IsZero(2*G) and not IsZero(4*G) and not IsZero(8*G)
        and not IsZero(r*G) and not IsZero(2*r*G) and not IsZero(4*r*G)
        and IsZero(n*G);
    "Monty Generator:", G;

    // find base point:
    x := F!0;
    repeat
        repeat
            x +:= F!1;
        until IsSquare(x^3 + A*x^2 + x);
        P := E![x, Sqrt(x^3 + A*x^2 + x)];
    until not IsZero(8*P) and IsZero(r*P);
    "Monty Base Point:", P;
end procedure;

function MontyCurve(m, A)
    p := 2^m;
    repeat
        p := PreviousPrime(p : Proof := false);
    until p mod 8 eq 5;
    delta := 2^m - p;
    if delta gt m then
        return false;
    end if;
    F := GF(p);
    sgnA := (A lt 0) select "- "*Sprint(-A) else "+ "*Sprint(A);
    z := F!2;
    assert not IsSquare(z);
    if IsSquare(F!A - 2) or IsSquare(F!A^2 - 4) then
        return false;
    end if;
    // NB: now (A - 2)/(A + 2) is not a square either
    // check curve y^2 = x^3 + A*x^2 + x:
    ok, E := IsEllipticCurve([0, F!A, 0, 1, 0]);
    if not ok then
        return false;
    end if;
    n := #E;
    if (n mod 8 ne 0) or not IsProbablePrime(n div 8) then
        return false;
    end if;
    r := n div 8;
    if r lt 2^(m - 3) then
        return false;
    end if;
    // check twist v^2 = u^3 + A*z*u^2 + z^2*u:
    ok, Et := IsEllipticCurve([0, A*z, 0, z^2, 0]);
    if not ok then
        return false;
    end if;
    nt := #Et;
    if (nt mod 4 ne 0) or not IsProbablePrime(nt div 4) then
        return false;
    end if;
    rt := nt div 4;
    if rt lt 2^(m - 3) then
        return false;
    end if;
    t := p + 1 - n;
    if nt ne p + 1 + t then
        return false;
    end if;
    r := n div 8; // "|r| =", Round(Log(2, r));
    sec := Log(2, Sqrt(Pi(RealField())*r/4));
    "Good Elligator 2 curve: y^2 = x^3 " * sgnA * "*x^2 + x",
    "over GF(2^" * Sprint(m) * " - " * Sprint(delta) *")",
    "at sec level 2^" * Sprint(sec),
    " with r =", r;
    assert IsProbablePrime(r);
    return true;
end function;
```

```
procedure MontyTests()
    "================";
    m := 159; A := 197782;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 191; A := -281742;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 221; A := 117050;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 255; A := 486662;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 383; A := 2065150;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 511; A := 530438;
    if MontyCurve(m, A) then
        MontyGenBasept(m, A);
    else
        "LOGIC ERROR!";
    end if;
    "================";
end procedure;
```

```
MontyTests();

procedure EddieGenBasept(m, d)
    p := 2^m;
    repeat
        p := PreviousPrime(p : Proof := false);
    until p mod 4 eq 3;
    F := GF(p);
    d := F!d;
    E := EllipticCurve([0, 2*d + 2, 0, (1 - d)^2, 0]);
    n := #E;
    r := n div 4;
    assert n mod 4 eq 0 and IsProbablePrime(r);

    // find generator:
    y := F!0;
    repeat
        repeat
            y +:= F!1;
            U := (1 - d)*(1 - y)/(1 + y);
            ok, V := IsSquare(U^3 + (2*d + 2)*U^2 + (1 - d)^2*U);
        until ok;
        G := E![U, V];
    until not IsZero(2*G) and not IsZero(4*G)
        and not IsZero(r*G) and not IsZero(2*r*G)
        and IsZero(n*G);
    x := 2*U/V;
    assert x^2 + y^2 eq 1 + d*x^2*y^2;
    "Eddie Generator: (", x, ",", y, ")";

    // find base point:
    y := F!0;
    repeat
        repeat
            y +:= F!1;
            U := (1 - d)*(1 - y)/(1 + y);
            ok, V := IsSquare(U^3 + (2*d + 2)*U^2 + (1 - d)^2*U);
        until ok;
        P := E![U, V];
    until not IsZero(4*P) and IsZero(r*P);
    x := 2*U/V;
    assert x^2 + y^2 eq 1 + d*x^2*y^2;
    "Eddie Base Point: (", x, ",", y, ")";
end procedure;
```

```
function EddieCurve(m, d)
    p := 2^m;
    repeat
        p := PreviousPrime(p);
    until p mod 4 eq 3;
    delta := 2^m - p;
    m0 := (m ne 168) select m else 2*m; // see discussion on Telex
    if delta gt m0 then
        return false;
    end if;
    F := GF(p);
    sgnd := (d lt 0) select "- "*Sprint(-d) else "+ "*Sprint(d);
    d := F!d;
    if IsSquare(d) /* not complete Edwards */ then
        return false;
    end if;
    if IsSquare(1 - d) /* trivial quad twist */ then
        return false;
    end if;
    assert IsSquare(-d);
    u := Sqrt(-d);
    if not IsSquare(2*(u - 1)/(u + 1)) /* injective map undefined */ then
        return false;
    end if;
    // check curve x^2 + y^2 = 1 + dx^2y^2,
    // or equivalently y^2 = x^3 + (2d + 2) x^2 + (1 - d)^2 x:
    ok, E := IsEllipticCurve([0, (2*d + 2), 0, (1 - d)^2, 0]);
    if not ok then
        return false;
    end if;
    n := SEA(E : AbortLevel := 2, MaxSmooth := 4);
    if (n mod 4 ne 0) or not IsProbablePrime(n div 4) then
        return false;
    end if;
    r := n div 4;
    if r lt 2^(m - 3) then
        return false;
    end if;
    // check twist x^2 + y^2 = 1 + (1/d)x^2y^2,
    // or equivalently v^2 = u^3 + 2(1 + d)/(1 - d)u^2 + u:
    ok, Et := IsEllipticCurve([0, 2*(1 + d)/(1 - d), 0, 1, 0]);
    if not ok then
        return false;
    end if;
    nt := #Et;
    if (nt mod 4 ne 0) or not IsProbablePrime(nt div 4) then
        return false;
    end if;
    rt := nt div 4;
    if rt lt 2^(m - 3) then
        return false;
    end if;
    t := p + 1 - n;
    if nt ne p + 1 + t then
        return false;
    end if;
    r := n div 4; // "|r| =", Round(Log(2, r));
    sec := Log(2, Sqrt(Pi(RealField())*r/4));
    "Good Elligator 1 curve: x^2 + y^2 = 1 " * sgnd * "*x^2*y^2",
    "over GF(2^" * Sprint(m) * " -  "* Sprint(delta) * ")",
    "at sec level 2^" * Sprint(sec),
    " with r =", r;
    assert IsProbablePrime(r);
    return true;
end function;

procedure EddieTests()
    "================";
    m := 157; d := -42000;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 168; d := -715;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 190; d := -15584;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 222; d := 160102;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 251; d := -1174;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 382; d := -67254;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
    m := 521; d := 376015;
    if EddieCurve(m, d) then
        EddieGenBasept(m, d);
    else
        "LOGIC ERROR!";
    end if;
    "================";
end procedure;

EddieTests();
```