OPEN SOURCE IS NOT ENOUGH

ATTACKING THE EC-PACKAGE OF BOUNCYCASTLE VERSION $1.x\_132$

DANIEL MALL AND QING ZHONG

Abstract. BouncyCastle is an open source Crypto provider written in Java which supplies classes for Elliptic Curve Cryptography (ECC). We have found a flaw in the class ECPoint resulting from an unhappy interaction of elementary algorithms. We show how to exploit this flaw to a real world attack, e.g., on the encryption scheme ECIES. BouncyCastle has since fixed this flaw (version 1.x_133 and higher) but all older versions remain highly vulnerable to an active attacker and the attack shows a certain vulnerability of the involved validation algorithms.

Key Words: ECC, BouncyCastle

1. Introduction

As already pointed out by other authors, e.g. [Ngu04], the availability of source code is not a guarantee for security. It is mandatory that the code will be scrutinized by various experts. Obviously, this cannot be the task of the provider only. The aim of this paper is to explain, firstly, a software flaw found about two years ago by the first author (cf. [Mal]) in the math.ec package of the open source crypto software BouncyCastle (cf. [Leg]), and, secondly, how one can exploit this flaw to a real world attack against Elliptic Curve Cryptography (ECC).
The flaw is by now fixed (version 1.x_133). But for those who haven't used the updated version their applications remain highly vulnerable. Furthermore, the attack shows a certain vulnerability of the involved validation algorithms.

2. The Flaw

The BouncyCastle package org.bouncycastle.math.ec version 1.x_132 consists of the following four classes:

(1) the class ECConstants which provides the numbers 0 and 1 as BigIntegers,
(2) the abstract class ECCurve which represents an elliptic curve in Weierstrass normal form,
(3) the abstract class ECFieldElements which represents an element in the Galois field over which the curve is defined, and
(4) the abstract class ECPoint which represents the points on the elliptic curve and implements the arithmetic of this curve.

All the abstract classes contain two non-abstract nested subclasses derived from them: .Fp and .F2m representing curves defined over $\mathbb{Z}_p$, $p > 2$, over fields of characteristic 2, respectively.
The flaw occurs in the class ECPoint.Fp and, hence, applies only to elliptic curves of characteristic $p > 2$. We assume in the following that there is given an elliptic curve in Weierstrass normal form over a Galois field of the form $\mathbb{Z}_p$, p a prime number:

$$E: \quad y^2 = x^3 + ax + b \tag{1}$$

It is well known that there are different laws for the addition of two points on an elliptic curve in dependence whether we add two different points or whether we double a single point.
Therefore, it is tempting to introduce two methods add() and twice() to implement these addition laws. BC does this without any contract what is certainly a dangerous idea. However, if you use the method add() to double a point, you will get a division by zero, i.e., in Java an Arithmetic Exception. This occurs also, if we try to add points the sum of which equals the point at infinity, i.e., the neutral element of the underlying group. So, one has to enforce that the method add() is never used to double a point.
The problem is now that the method multiply() in the class ECPoint.Fp uses a NAF-representation (cf. [HMV77] p.98) of the multiplier:

```
        public ECPoint multiply(BigInteger k)
        {
            // BigInteger e = k.mod(n); // n == order this
            BigInteger e = k;

            BigInteger h = e.multiply(BigInteger.valueOf(3));

            ECPoint R = this;

            for (int i = h.bitLength() - 2; i > 0; i--)
            {
                R = R.twice();

                if (h.testBit(i) && !e.testBit(i))
                {
                    //System.out.print("+");
                    R = R.add(this);
                }
                else if (!h.testBit(i) && e.testBit(i))
                {
                    //System.out.print("-");
                    R = R.subtract(this);
                }
                // else
                // System.out.print(".");
            }
            // System.out.println();

            return R;
        }
    }
```

This algorithm is similar to the well-known square-and-multiply algorithm to add k times a given point P, i.e., in this situation it is rather a double-and-add algorithm. However the above NAF-algorithm uses beside doubling and addition also subtractions which results sometimes in fewer operations. We demonstrate this in the following example.

Example 2.1. *Computation of* $31 * P$ *according to the double-and-add algorithm:*

$$2 * P = P + P, \quad 3 * P = P + 2 * P, \quad 6 * P = 2 * (3 * P), \quad 7 * P = P + 6 * P,$$

$$14 * P = 2 * (7 * P), \quad 15 * P = P + 14 * P, \quad 30 * P = 2 * (15 * P), \quad 31 * P = P + 30 * P \ .$$

We need 8 *operations.*

Computation of $31 * P$ *according to the NAF-algorithm:*

$$2 * P = P + P, \quad 4 * P = 2 * P + 2 * P, \quad 8 * P = 4 * P + 4 * P,$$

$$16 * P = 8 * P + 8 * P, \quad 32 * P = 16 * P + 16 * P, \quad 31 * P = 32 * P - P \ .$$

We need 6 *operations.*

Remark 2.2. *The behaviour of the NAF-algorithm in Example 2.1 is typical: For any* $n \equiv 3$ *mod* 4*, the calculation of* $n * P$ *ends with a subtraction.*

Now, we assume that we have an elliptic curve with a point P of order $ord(P) \equiv 1 \bmod 4$. If we calculate $(ord(P) - 2) * P$ with the multiply-method, we end up calculating $(ord(P) - 2) * P = (ord(P) - 1) * P + (-P)$. Since $(ord(P) - 1) * P = (-P)$, we have to calculate $(-P) + (-P)$. In this situation the method multiply() selects the method add() and throws an ArithmeticException. An ArithmeticException means that a division by zero occurred, which is interpreted that we have reached the point at infinity. Hence, we can pretend that the order of the point P is $ord(P) - 2$.

Example 2.3. *Given the elliptic curve* $E: \ y^2 = x^3 + 4x + 20$ *over the field* $\mathbb{Z}_{29}$*. The associated group has order* 37*. Take a point on* E*, e.g.,* $P = (13, 6)$*. Obviously, we have* $ord(P) = 37 \equiv 1$ *mod* 4*. Hence, if BouncyCastle calculates* $35 * P$ *it throws an ArithmeticException.*

3. The Attack

We explain now how one can exploit the above flaw to an attack on ECC. To fix ideas, we assume an environment which uses ECC (cf., e.g., [HMV77] p. 189) to encrypt data with a Crypto provider which is contaminated with the above flaw, e.g., the old versions 1.x_132 of BouncyCastle.
We want to use the algorithms of the standard FIPS 186-2. A good reference is the book [HMV77]. Our aim is to construct domain parameters $D = (q, FR, S, a, b, P, n, h)$ for ECC (cf., e.g., [HMV77] p.172) with q a prime chosen at random according to FIPS 186-2, appendix 4, which passes the validation algorithms (cf. FIPS 186-2, appendix 5, or [HMV77] p.175) and can be used to encrypt and decrypt data but the data are completely insecure.
We first select a prime number p with an appropriate bit length. Then we start the following algorithm beginning by choosing an elliptic curve over p at random.

Algorithm 3.1.

(1) *Select* $a, b \in \mathbb{Z}_p$ *according to FIPS 186-2, appendix 4.*
(2) *Compute* $N = |E(\mathbb{Z}_p)|$.
(3) *Verify that* $N - 2$ *is a prime number* $\neq p$*. If not, then go to step 1.*
(4) *Verify that* $N - 2$ *does not divide* $p^k - 1$ *for* $1 \leq k \leq 20$*. If not, then go to step 1.*
(5) *Verify that* $N = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ *with primes* p_i *with a bit length of about* l *bits, e.g.,* 50 *bits (cf. below). If not, then go to step 1.*
(6) *Set* $h \longleftarrow 1$.
(7) *Select an arbitrary point* $P \in E(\mathbb{Z}_p)$ *with a large period, e.g.,* N.
(8) *Return*$(p, \ FR, \ S, \ a, \ b, \ P, \ N - 2, \ h)$.

To fix ideas, we assume that Alice and Bob want to communicate together via the ECIES scheme (cf. [HMV77] p.189). Assume further that the active attacker Oscar presents domain parameters generated with Algorithm 3.1 to Alice and Bob. Alice and Bob will evaluate the parameters with, e.g., the *Explicit Domain Parameter Validation* (cf. Algorithm 4.15 in [HMV77] p.175). Because of the steps 3, 4 and 6 of algorithm 3.1 these domain parameters pass the validation. Alice and Bob will select their private and public keys, (d_A, Q_A) and (d_B, Q_B). To encrypt, Bob performs the following steps (cf. [HMV77] Algorithm 4.42) with the public key Q of Alice. (In the following n stands for $N-2$.)

(1) Selection of $k \in [1, n-1]$.
(2) Computation of $R = k * P$ and $Z = h * k * Q$ $(= k * Q)$. If $Z = \infty$ then go to step 1.
(3) Derivation of the keys k_1, k_2 with the x-coordinate of Z, the point R and an appropriate key derivation function.

Observe that in the calculation of Z the true period of P comes in with a very high probability, because in general $kd \geq n + 2 = N$.
To decrypt, Alice performs the following steps (cf. [HMV77] Algorithm 4.43) with her private key d.

(1) Validation of R.
(2) Computation of $\tilde{Z} = d * R$. If $Z = \infty$ then the algorithm fails.
(3) With the x-coordinate of Z, the point R and an appropriate key derivation function, Alice derives the keys k_1, k_2.

This works because

$$\tilde{Z} = d * R = d * (k * P) = k * (d * P) = k * Q = Z$$

and the validity of these equalities is not affected by the wrong period.
Since $N - 2$ is a prime, Alice and Bob will have no problem with encryption and decryption. On the other hand, Oscar is able to solve the discrete logarithm problem, e.g., $Q_A = d_A P$ with the Pohlig-Hellman and Pollard-rho-algorithms, since he has chosen the bit length l of the prime factors of N appropriately.
The following is a toy example. In section 4 we will give a realistic one.

Example 3.2. *As prime number we choose* $p := 48611 \equiv 3$ *mod* 4 *and our elliptic curve is*

$$E(\mathbb{Z}_{48611}): \ y^2 = x^3 - 3x + 26221 \quad . \tag{2}$$

Point counting gives $N := |E(\mathbb{Z}_{48611})| = 48493 = 71 \cdot 683$ *and* $N - 2$ *is a prime number.*

4. A realistic Example

In this section, we want to present a realistic example of key length about 160 bits.
First, we generate a prime number p with $p \equiv 3 \bmod 4$ of at least 160 bit length and a 160 bit seed S.
Choosing the curve parameter $a = -3$ we get an appropriate curve by algorithm 3.1. Here is an example:

Example 4.1.

Fake-162: $p = 2^{161} + 107$, $a = -3$, $h = 1$

$$\begin{array}{rcl} S & = & 0x\ 80272944CD96A014B660040E91108B00FB8566B8 \\ r & = & 0x\ A31C83EEF1C7F13197CE2EE60F42D0D5AB5F5FA6 \\ b & = & 0x\ 67CF5897EB5790B8B7DC3020425DD4F34E49A64F \\ n & = & 0x\ 1FFFFFFFFFFFFFFFFFFFFFFC42EAC6771DFD41699F5 \\ x & = & 0x\ 135954F24690DBB44D0D31A850EBEB8FA2E93E990 \\ y & = & 0x\ 92DFC232BD02D83D58B19647F505C7AF44982839 \end{array} \tag{3}$$

The length of p is 162 *bits and the group is cyclic of order*

$$N = 29230032746618058364073693829515503178270663377 81 .$$

The number $n := N - 2$ is prime and the factorization of N is

$$N = 87299 \cdot 1447868275134167891 \cdot 23125491975274153080658909 .$$

The point $P := (x, y)$ has order N.
It is easy to check that the above domain parameters pass all the checks of the validation algorithm (cf. [HMV77] *Algorithm 4.15) provided that the test software for the curve validation is contaminated with the bug of the bouncycastle software version 1.x_132. Otherwise we would get $nP \neq \infty$.*

To solve an ECDLP problem for the above curve with Pollard's ρ-algorithm needs about $0.5\ 10^{13}$ *group operations which is feasible.*

Remark 4.2. *Curves with the wanted properties are easy to find. But most of them have a small first prime factor, e.g. $p_1 = 3$.*
That the number n doesn't appear random isn't a coincidence. It follows from the special form of the prime p and the theorem of Hasse (cf [Sil86] *p.131)*

$$|n + 2 - p - 1| \leq 2\sqrt{p} .$$

Remark 4.3. *The calculations of example 4.1 were done with the EC package of the software LiDIA-2.2 (cf.* [LiD]*).*

5. A Denial-of-Service Attack on ECDSA

The attack desribed in section 3 reveals a weakness in ECC encryption schemes like ECIES. ECC signature algorithms like ECDSA are more resilient against our attack. The ECDSA signature algorithm does the following steps (cf., e.g., [HMV77] Algorithm 4.29) given domain parameters $D = (q, FR, S, a, b, P, n, h)$:

(1) Select $k \in [1, n - 1]$
(2) Compute $kP = (x, y)$ and convert x to an integer $\tilde{x}$.
(3) Let r be the remainder of $\tilde{x}$ mod n. If $r = 0$ go to step 1.
(4) Compute $e := H(m)$.
(5) Compute $s := (e + dr)k^{-1}$ mod n. If $s = 0$ go to step 1.
(6) Return (r, s).

In the verification algorithm (cf., e.g., [HMV77] Algorithm 4.30), we have to calculate

(1) Compute $e := H(m)$.
(2) Compute $w := s^{-1}$ mod n.
(3) Compute $u_1 := ew$ mod n and $u_2 := rw$ mod n.
(4) Compute $X := u_1P + u_2Q$.
(5) Compute $\tilde{x}$ the x-coordinate of X mod n.

Now $X := u_1P + u_2Q$ equals $X := (u_1 + u_2d)P = kP$. In general, $u_1 + u_2d$ is bigger than n. Hence the last equality is only correct modulo the the true period, i.e, it fails with very high probability. This gives a opportunity for a DoS Attack: Assume a system with the described flaw installed. If you are able to bring in a faked elliptic curve, e.g. as a reserve parameter, then after, e.g., a provoked revocation of the parameter in use, the reserve parameter comes in and no signature verification is possible anymore. It will take certainly some time to find such a flaw.

6. Conclusions

Since the counting of points on a given elliptic curve E over a prime p is in general a time consuming task, the validation algorithm for elliptic curves omits it (cf. [HMV77] p.175). This implies that our attack is a real threat in situations where domain parameters are only tested against the above cited validation algorithm. This includes situations where, e.g., the involved persons possess only limited knowledge or where a lot of individual elliptic curves for nearly real time applications are used.

References

[HMV77] Darrel Hankerson, Alfred Menezes, and Scott Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 1977.

[Leg] Legion of the BouncyCastle, *Bouncycastle*, www.bouncycastle.org.

[LiD] LiDIA , *A Library for Computational Number Theory*, www.cdc.informatik.tu-darmstadt.de/TI/LiDIA.

[Mal] D. Mall, *Bug in class ECPoint.Fp*, www.bouncycastle.org/devmailarchive , 13 apr 2006.

[Ngu04] Phong Nguyen, *Can we Trust Cryptographic Software? Cryptographical Flaws in GNU Privacy Guard v1.2.3*, Proc. EUROCRYPT '04 (Berlin) (Ch. Cachin and J. Camenisch, eds.), LNCS, vol. 3027, Springer, 2004, Advances in Cryptology EUROCRYPT 2004, pp. 555–570.

[Sil86] J. H. Silverman, *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, vol. 106, Springer, New York, 1986.

VBS, Sektion Kryptologie, CH-3003 Bern and FHNW, CH-5210 Windisch.

FHNW, CH-5210 Windisch.