Hyperbola

Dcoder

1 Introduction

Hyperbola [6] is the latest challenge brought to us by nwert. The scheme is straightforward: it is a Nyberg-Rueppel signature [7] defined over the group of solutions (x, y) to the equation

$$x^2 - dy^2 = 1 \pmod p,$$

where $p = 16831744095843413413$ and $d = 6731525366519611944$. The base point and public key are, respectively,

$$\begin{aligned} P &= (12879847090741109435, 11609566893283354662) \\ Q &= (3839438711590932798,\ 9101041563847465056). \end{aligned}$$

The group addition is given by

$$(x_3, y_3) = (x_1x_2 + dy_1y_2, x_1y_2 + x_2y_1),$$

along with the identity element $\mathcal{O} = (1, 0)$.

This curve is often known as the Pell conic [4, 5], named after the related Pell equation [3]. This is not an elliptic curve; its genus is 0 and we have two possible nontrivial cases:

- $d = a^2$ is a square modulo p, in which case the equation factors as $(x-ay)(x+ay) = 1$, and the group of solutions is isomorphic to $\mathbb{F}_p^\times$, with order $p - 1$.
- d is not a square modulo p, in which case the group is isomorphic to the norm-1 subgroup of $\mathbb{F}_{p^2}^\times$, which has order $p + 1$.

The case here is the latter. We can solve the logarithm of P and Q directly, using rho, or send the points to $\mathbb{F}_{p^2}^\times$ and solve there. While there are asymptotically fast algorithms to solve logarithms in $\mathbb{F}_{p^2}^\times$ [1, 2], it is much quicker and easier to solve the logarithm directly in the curve using parallel rho [8, 9], particularly given such a simple addition rule. The complexity of solving it this way is around $\sqrt{\pi(p + 1)} \approx 2^{32}$ point additions, which is fairly quick.

There is not much more to say about this challenge. There is one bug in the signature verification that limits the amount of valid signatures accepted to about half. The verification process for Nyberg-Rueppel goes like this:

```
def verify(name, signature):
  h = int(sha1(name).hexdigest()[0:16], 16)
  c = int(signature[0:16], 16)
  d = int(signature[17:], 16)
  x, _ = point_add(point_mul(P, d), point_mul(Q, c))
  return h % n == (c - x) % n
```

Since x is computed modulo p, it is larger than the order $n = (p+1)/2$ about half the time. Since it is not reduced modulo n prior to the subtraction with c, the modular reduction fails and the signature is declared invalid. Therefore we must ensure that the x coordinate of $dP + cQ$ is smaller than n to work around this issue.

References

[1] Barbulescu, Razvan and Cécile Pierrot: *The Multiple Number Field Sieve for Medium and High Characteristic Finite Fields.* http://hal.inria.fr/hal-00952610, February 2014.

[2] Gamal, Taher El: *A subexponential-time algorithm for computing discrete logarithms over GF(p^2).* IEEE Transactions on Information Theory, 31(4):473–481, 1985.

[3] Hendrik W. Lenstra, Jr.: *Solving the Pell equation.* Volume 44 of *Mathematical Sciences Research Institute Publications*, pages 1–23. Cambridge University Press, Cambridge, 2008, ISBN 978-0-521-80854-5. http://library.msri.org/books/Book44/index.html.

[4] Lemmermeyer, F.: *Conics — a Poor Man's Elliptic Curves.* ArXiv Mathematics e-prints, November 2003. http://arxiv.org/abs/math/0311306.

[5] Menezes, Alfred and Scott A. Vanstone: *A note on cyclic groups, finite fields, and the discrete logarithm problem.* Appl. Algebra Eng. Commun. Comput., 3:67–74, 1992.

[6] nwert: *hyperbola.* http://crackmes.de/users/nwert/hyperbola/, June 2014.

[7] Nyberg, Kaisa and Rainer A. Rueppel: *A New Signature Scheme Based on the DSA Giving Message Recovery.* In Denning, Dorothy E., Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (editors): *ACM Conference on Computer and Communications Security*, pages 58–61. ACM, 1993, ISBN 0-89791-629-8.

[8] Oorschot, Paul C. van and Michael J. Wiener: *Parallel Collision Search with Application to Hash Functions and Discrete Logarithms.* In Denning, Dorothy E., Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu (editors): *ACM Conference on Computer and Communications Security*, pages 210–218. ACM, 1994, ISBN 0-89791-732-4.

[9] Oorschot, Paul C. van and Michael J. Wiener: *Parallel collision search with cryptanalytic applications.* J. Cryptology, 12(1):1–28, 1999.

A Solution code

```python
import sys
from hashlib import sha1
from random import randint

p = 0xE9965E13A7066DA5
d = 0x5D6B2EA7DBEEC228

P = (0xB2BE68B04CF8E2BB, 0xA11D77944E4A2826)
Q = (0x35486D2A7D42D13E, 0x7E4D65153B319860)
n = 0x74CB2F09D38336D3
x = 4300377673800084310

def point_add(P, Q):
  x1, y1 = P
  x2, y2 = Q
  return ((x1*x2 + d*y1*y2)%p, (x1*y2 + x2*y1)%p)

def point_mul(P, e):
  R = (1, 0)
  m = 1 << 64
  while m != 0:
    R = point_add(R, R)
    if m & e != 0:
      R = point_add(R, P)
    m >>= 1
  return R

def verify(name, signature):
  h = int(sha1(name).hexdigest()[0:16], 16)
  c = int(signature[0:16], 16)
  d = int(signature[17:], 16)
  x, _ = point_add(point_mul(P, d), point_mul(Q, c))
  return h % n == (c - x) % n

def sign(name):
  h = int(sha1(name).hexdigest()[0:16], 16)
  while True:
    u = randint(1, n-1)
    c = (point_mul(P, u)[0] + h) % n
    d = (u - x*c)%n
    if point_mul(P, (d + x*c)%n)[0] < n: # work around bug
      break
  return "%016X-%016X" % (c, d)

if len(sys.argv) != 2:
  print "Usage: %s <name>" % sys.argv[0]
  sys.exit(1)

s = sign(sys.argv[1])
assert( verify(sys.argv[1], s) )
print s
```