**Problem**

In the ElGamal schemes (for both encryption and signatures), g is supposed to be the generator of the entire Z^\*\_p group.

However, in the current implementation, g is more simply the generator of a random sub-group of Z^\*\_p.

The order of such sub-group may be smaller than p-1, and since there are not constraints or checks on the factorization of p-1, the order may be \*much\* smaller than what it should be.

To say, if I limit the bit size to 8 bits, I get p=211 and g=107. The order of g is 42, much less than the expected (and "secure") 210!

<https://www.rfc-editor.org/rfc/rfc2785>

* Do you know if there is a way to take an existing key and find out whether it has this problem? (Even better, to compute the order of the sub-group?)
  + One way is to factor p-1 and find all its divisors. The divisors are also all the possible orders a sub-group of Zp can have. The generator g is safe (with respect to this problem) if **g^d mod p is not 1 for all divisors other than p-1**. **That may not be enough though**: **when the key is used for signing**, one should also check that **g and g^{-1} don't divide p-1** (see patch). Those are at least the additional constraints I have found so far for the key...
* Also, given that this is a \*random\* sub-group, does this actually help an attacker? I'm going to fix this anyway, but I want to get an understanding of how bad this is. I don't know much about ElGamal encryption.
  + g is also made public, so that would be only relevant if the attack was based on some precomputation I think. The main problem I see is that the signature space (when the key is used for signing) or the public key space (when the key is used for encryption) may be greatly reduced from log(p) bits down to up 1 bit (worst case if the order of g is 2). That's already bad enough, but it could be that such fact may also be used to work out the private key (which the RFC you have found seem to imply).
* Is it okay if I just merge your entire tree (up to 6f312637208cc70b231892109564e2aebc92728a)?
  + I can submit a complete/tested/polished set of changes in one week time. As you can guess, the changes are mostly about improving documentation of the public key module: right now, qnew and maybe pycrypt.rst are still missing. I run into this problem by reviewing the Elgamal code.

Text, letter

Description automatically generated

Le problème est réellement dans cette partie du fichier :

**ElGamal.py**

Graphical user interface, text, application, email

Description automatically generated

**Fix**

**ElGamal.py**

while 1:

# We must avoid g=2 because of Bleichenbacher's attack described

# in "Generating ElGamal signatures without knowning the secret key",

# 1996

#

obj.g = number.getRandomRange(3, obj.p, randfunc)

safe = 1

**if pow(obj.g, 2, obj.p)==1:**

safe=0

**if safe and pow(obj.g, q, obj.p)==1:**

safe=0

# Discard g if it divides p-1 because of the attack described

# in Note 11.67 (iii) in HAC

**if safe and divmod(obj.p-1, obj.g)[1]==0: (spécifiquement ici)**

safe=0

# g^{-1} must not divide p-1 because of Khadir's attack

# described in "Conditions of the generator for forging ElGamal

# signature", 2011 <https://www.researchgate.net/publication/234113996_Conditions_on_the_generator_for_forging_ElGamal_signature>

ginv = number.inverse(obj.g, obj.p)

**if safe and divmod(obj.p-1, ginv)[1]==0: (et ici)**

safe=0

if safe:

break