Project # 2

This project is due Thursday, February 19.

1. *Efficient Modular Exponentiation*

   Implement different versions of the modular exponentiation `my_pow(x,e,n)` such that the returned value y is $y = x^e \mod n$. Compare the performance of your implementation to the built-in Python function `pow(x,e,n)` as well as to each other.

   (a) Implement `my_pow_SqMul(x,e,n)` using the square and multiply algorithm.

   (b) Implement `my_pow_SlWin(x,e,n)` using the sliding window method. Test your code for window size $k = 4$.

2. *RSA for Encryption and Decryption*

   Implement RSA encryption and decryption as described in PKCS #1, Version 2.1. Use the provided template for your implementation. A detailed description can be found in the standard: `http://tools.ietf.org/html/rfc3447`

   We use SHA-256 as hash function for computing the digest of L (L will be empty by default) as well as for the mask generation function MGF. You may use `hashlib.sha256` as implementation of SHA-256 and `os.urandom` to generate random bytes. Test vectors are provided in the template.

3. *Fault analysis of RSA-CRT*

   The goal of this exercise is to write python functions that recover the factorization of the public modulus N and the secret key d of a signature engine employing RSA-CRT.

   (a) write a function `Bellcore(sig,sig_p,N)` which returns the factorization of $N = pq$, given the public modulus N, a valid RSA signature $\sigma = m^d \mod N$ and a faulty signature $\sigma_p = \sigma'$ that has been obtained by inducing a fault during the signature generation using RSA-CRT.

   (b) write a function `Lenstra(m,sig_p,e,N)` which returns the factorization of $N = pq$, given the public key $\langle N, e\rangle$ and a faulty signature $\sigma_p = \sigma'$ of a message m.

   To test your functions, recover the private key $sk = \langle d, p, q\rangle$ for the following parameters:

$N =$
277300785103520926588582123255714081202079691180626668720378057977823550
365895334530032934470837111774308827069293647074506100770136544876408462
935663524635250182016948244643971476982514603624475948898274106699510799
199242248247504578150499994878162024532535355724647778297500482316078569
000202053036651904 29

$e =$
274045216667441409790619393955475845125428899490343739531016779904995128
248277909472237683916610048380645305311989144767180235946627955041071142
965066782126237354625151032456942473978410919493246652003115680868462448
421174389747350960475502007114457368959336383183420961549139084558081968
68575997827586823873

Bellcore: $c =$
110331116750664335997317756414034593676827674292521451836660666517636943
067873187531427905212473824895936446866704529539621307525708186984572180
712836042015241200290689584141263049129439294410401242805039817756123406
285689783253031964909728208382949124557180676658649620225433967304674723
60766384623334635574

$c' =$
197969236886912254302487358920402549270612987516437634605708315677609307
379217861544761129301243491569410999085770858853823543105736578158155122
458453559113883562217821540774638296040145572078790463883273647443470216
839689546058724888102141194520728908281102512542549079976518449982427852
55936647330134950937

Lenstra: $m =$
555112700422434673254004144713045961271996434154724461486708153068199909
875185722713550235322582072957336801127546164227058333266312722265726685
138636425451996244796698439318493152708314210196742905142676215629975706
488896098353980024997379282156513674480266410172000272870332435712335330
1041293846831380875

$c' =$
503618350997341354798184783492595408556664556830258915268020300917325330
422446650991154431747248121844949763553222460288858034530084155982857478
5126853457

Good luck and have fun!