### From repeated nonce, recover DSA nonce

You should know how to find a collection of DSA-signed messages form this file [DSA nonce.txt](DSA%20Nonce.txt).  (NB: each message has a trailing space.)

These were signed under the following public key:

y = 2d026f4bf30195ede3a088da85e398ef869611d0f68f0713d51c9c1a3a26c95105d915e2d8cdf26d056b86b8a7b85519b1c23cc3ecdc6062650462e3063bd179c2a6581519f674a61f1d89a1fff27171ebc1b93d4dc57bceb7ae2430f98a6a4d83d8279ee65d71c1203d2c96d65ebbf7cce9d32971c3de5084cce04a2e147821

It should not be hard to find the messages for which we have accidentally used a repeated "k". Given a pair of such messages, you can discover the "k" we used with the following formula:

(m1 - m2)

k = --------- mod q

(s1 - s2)

***Question 1*** Give the proof of this formula (paper work).

If you want to demystify this, work out that equation from the original DSA equations.

**()**

Remember all this math is mod q; s2 may be larger than s1, for instance, which isn't a problem if you're doing the subtraction mod q. (And don't forget that modular inverse function!)

***Question 2*** (Optional): Now, try to recover *k* , submit it.

What's my private key? Its SHA-1 (from hex) is:

ca8f6f7c66fa362d40760d135b763eb8527d3d52

A brief introduction to DSA

(<https://en.wikipedia.org/wiki/Digital_Signature_Algorithm> )

## Signing

Let H be the hashing function and m the message:

* Generate a random per-message value k where 0 < k < q
* Calculate r=\left(g^{k}\bmod\,p\right)\bmod\,q
* In the unlikely case that r=0, start again with a different random k
* Calculate s=k^{-1}\left(H\left(m\right)+xr\right)\bmod\,q
* In the unlikely case that s=0, start again with a different random k
* The signature is \left(r,s\right)

The first two steps amount to creating a new per-message key. The modular exponentiation here is the most computationally expensive part of the signing operation, and it may be computed before the message hash is known. The modular inverse k^{-1}\bmod\,q is the second most expensive part, and it may also be computed before the message hash is known. It may be computed using the extended Euclidean algorithm or using Fermat's little theorem as k^{q-2}\bmod\,q.

## Verifying

* Reject the signature if 0<r<q or 0<s<q is not satisfied.
* Calculate w=s^{-1}\bmod\,q
* Calculate u_{1}=H\left(m\right)\cdot w\,\bmod\,q
* Calculate u_{2}=r\cdot w\,\bmod\,q
* Calculate v=\left(\left(g^{u_{1}}y^{u_{2}}\right)\bmod\,p\right)\bmod\,q
* The signature is invalid unless v=r

DSA is similar to the ElGamal signature scheme.

http://www.cryptopals.com/sets/6/challenges/43  
http://www.cryptopals.com/sets/6/challenges/44