**Lab 8. MASH**

Ref. Appendix B, Textbook

**1. Example.** The following is an example of the MASH (*Modular Arithmetic Secure Hash*) hash function in Sage. MASH is a function based on the use of modular arithmetic. It involves use of an RSA-like modulus M, whose bitlength affects the security. M should be difficult to factor, and for M of unknown factorization, the security is based in part on the difficulty of extracting modular roots. M also determines the block size for processing messages. In essence, MASH is defined as:

*Hi* = ((*xi* + *Hi*–1) 2 OR *Hi*–1) (mod *M*)

Where

*H*i-1 = The largest prime less than

*xi* = the *i*th digit of the base *M* expansion of input *n*. So:

*n = x*0 + *x*1*M* + *x*2*M*2 + …

The following is an example of the MASH hash function in Sage

#

# This function generates a mash modulus

# takes a bit length, and returns a Mash

# modulus l or l-1 bits long (if n is odd)

# returns p, q, and the product N

#

def generate\_mash\_modulus(l):

m = l.quo\_rem(2)[0]

p = 1

while (p < 2^(m-1)):

p = random\_prime(2^m)

q = 1

while (q < 2^(m-1)):

q = random\_prime(2^m)

N = p\*q

return (N, p, q)

#

# Mash Hash

# the value n is the data to be hashed.

# the value N is the modulus

# Returns the hash value.

#

def MASH(n, N):

H = previous\_prime(N)

q = n

while (0 != q):

(q, a) = q.quo\_rem(N)

H = ((H+a)^2 + H) % N

return H

The output of these functions running;

sage: data = ZZ(randint(1,2^1000))

sage: (N, p, q) = generate\_mash\_modulus(20)

sage: MASH(data, N)

220874

sage: (N, p, q) = generate\_mash\_modulus(50)

sage: MASH(data, N)

455794413217080

sage: (N, p, q) = generate\_mash\_modulus(100)

sage: MASH(data, N)

268864504538508517754648285037

sage: data = ZZ(randint(1,2^1000))

sage: MASH(data, N)

236862581074736881919296071248

sage: data = ZZ(randint(1,2^1000))

sage: MASH(data, N)

395463068716770866931052945515

**2.** Computational Example. The following describes the simple hash function:

Choose p, q primes and compute N = pq.

Choose g relatively prime to N and less than N.

Then a number n is hashed as follows:

H = gn mod N

If there is an m that hashes to the same value as n, then

gm ≡ gn mod N

so

gm-n ≡ 1 mod N

which implies that

m –n ≡ 0 mod φ (N)

So breaking this amounts to finding a multiple of φ (N), which is the hard problem in RSA.

1. Write a function that takes a bitlenght n and generates a modulus N of bitlength n and g less than N and relatively prime to it.
2. Show the output of your function from part (a) for a few outputs.
3. Using N, g, n as arguements write a function to perform the hashing.
4. For parts (d)-(f) compute the simple hash:
5. N = 600107, g = 154835, n = 239715
6. N = 548155966307, g = 189830397891, n = 44344313866
7. N = 604766153, g = 12075635, n = 443096843
8. Write a function that creates a collision given p and q. Show that your function works for a couple of examples.

Answer: