The RSA Encryption Scheme

Jean-Sébastien Coron

University of Luxembourg

1 Implementation of RSA

Install the Sage library, available at `http://www.sagemath.org/`. Alternatively, you can use the Sage Cell Server at `https://sagecell.sagemath.org`. Please submit a `.ipynb` file.

1. Implement the RSA key generation. You can use the function random_prime to generate a large prime, and the function ZZ.random_element to generate a random integer. You can use the inverse_mod function to compute a modular inverse. The function keyGen takes as input the bitsize n of the RSA modulus N.

```
def keyGen(n=512):
  "Generates an RSA key"
  return Nn,p,q,e,d
```

2. Implement the plain RSA encryption and decryption algorithms. You can use the powermod function. Check that decryption works on a random message.

```
def encrypt(m,Nn,e):
  pass
def decrypt(c,Nn,d):
  pass

def checkEnc():
  Nn,p,q,e,d=keyGen()
  m=ZZ.random_element(Nn)
  assert(decrypt(encrypt(m,Nn,e),Nn,d)==m)
```

3. Implement the RSA signature scheme with signature $\sigma = H(m)^d \bmod N$, where the output size of the hash function H is the same as the bit size of N, minus 4 bits. For this, we can concatenate the evaluation of a hash function h (for example, SHA-1), using an index for the message, truncating the last block:

$$H(m) = h(m\|0) \,\|\, h(m\|1) \|\cdots\| \, h(m\|k)$$

For the SHA-1 hash function, we use:

```
import hashlib

def sha1(m):
  h=hashlib.sha1()
  h.update(m.encode("utf-8"))
  return h.hexdigest()
```

We use the function Integer(hd,base=16) to convert an hexadecimal digest hd into an integer. The fullHash function below takes as input the message m to be signed (a string), and the length of the modulus. This length can be obtained using Nn.nbits().

```
# lN is the length of the modulus in bits
def fullHash(m,lN):
  k=ceil(lN/160)
  hf="".join(sha1(m+str(i)) for i in range(k))
  hf=hf[:lN//4-1]
  return Integer(hf,base=16)
```

Implement the signature generation and verification, and check that signature verification works. The signature algorithm should compute $\sigma = H(m)^d \bmod N$.

```
def sign(m,Nn,d):
  pass
def verify(s,m,Nn,e):
  pass

def checkSig():
  Nn,p,q,e,d=keyGen()
  m="message"
  assert(verify(sign(m,Nn,d),m,Nn,e))
```