Session 2

How RSA works

We have four main steps:

1. Key Generation
2. Key Distribution
3. Encryption
4. Decryption

[1] Key Generation:

Step 1: Alice chooses two large prime numbers p and q

Step 2: Alice calculates N and phi
N = p.q
phi = (p-1).(q-1)

Step 3: Alice Selects encryption exponent e: e is relatively prime to n.

Step 4: Alice calculates decryption exponent d:

$d \times e \equiv 1 \mod n$
Which is same as:
$d = e^{-1} \mod n$

Step 5: The keys are as follows:

Alice(K_U) = (e,N)
Alice(
K_R) = (d,N)

Note that Alice also keeps private, p, q and n.

Bob Repeats the above steps to generate his public and private keys.

Key Generation Example:

1. p= 5 and q=11
2. N = 5 x 11 = 55
3. phi = 4 x 10 = 40
4. e = 7
5. $d = 7^{-1} \mod 40 = 23$
6. K_U = (7,55)

   K_R = (23,55)

[2] Key Distribution:

Alice and Bob share their public keys over an insecure channel.

[3] Encryption:

If Bob wants to send X to Alice:

$y = x^e \mod N$ 1 < x < N

Encryption Example:

If Bob wants to send 'N' (N = No) to Alice.
Assume that we are using the same keys generated in the previous example.

x = 'N' = 13 (the base is the English alphabet)

$y = 13^7 \mod 55 \rightarrow y = 7$

Note that we are using e and m from the public key of Alice.

[4] Decryption:

If Alice receives Y from Bob, she can decrypt using:

$x = y^d \mod N \ 1 < x < N$

Decryption Example:

Using the above example:

y = 7

$x = 7^{23} \mod 55 \rightarrow x = 13$

Note that

we are using d and N from the Private key of Alice.

Assume we have a public key, how can we break RSA?

Easy, we have N and we know N = p.q so we just factorize N into p and q and find phi = (p-1)(q-1) and extract d thus breaking RSA.

Well…its easier said than done, this is exactly what we call **The Factorization Problem**!

Which happens to be one of those problems in the science of algorithms where if p and q were large enough (along with other properties) wasn't proven to have an efficient solution.

This drives us into the next section of this document

Problems with the factors and modulus p,q,N:

N is Prime

it's less complex to determine if a number is prime than to factor a number, phi would be equal to N - 1 and the entire cipher is broken

N is a Square

it’s also less complex to determine if a number is a perfect square than to factor a number, phi would be equal to N*(N-1) and the entire cipher is broken

P and Q are Close to Each Other (Fermat factoring)

Fermat's factorization method, also known as Fermat's factorization theorem, is one of the methods used to factorize integers. It's particularly effective when the difference between p and q is small. The basic idea behind Fermat's factorization method is to express NN as the difference of two squares:

$N=a2-b2=(a+b)(a-b)$

If NN can be expressed in this form, then pp and qq can be found by computing the greatest common divisor (GCD) of N with $(a+b)$ and $(a-b)$.

Therefore, if p and q are close to each other, an attacker can find them more easily using Fermat's factorization method or other similar techniques.

N Has Many Factors Not Just 2

THEORETICALLY, If N in RSA has many factors, not just two, it would complicate the factorization process and potentially enhance the security of the RSA encryption.

Surpisingly, that’s not really the case…

A fairly naive algorithm for factoring `N` is the following:

```
while N > 1:
  for p in increasing_primes:
    while p divides N:
      N = N / p
      factors.add(p)
```

With this algorithm,

340 282 366 920 938 463 463 374 607 431 768 211 456 can be factored in exactly 128 iterations of the innermost while. (The number, of course, is 2128)

The more prime factors a composite number has, the smaller those factors have to be. For example, 919·677=622 163. With the naive algorithm, this takes 157+1=158 iterations to factor. A number of roughly the same size comprised of three factors, 73·89·97=630 209, only takes 25+2=27 iterations to factor.

Similarly, a 1000-digit number composed of two roughly equally-sized factors will take about 10497 iterations to factor. A 1000-digit number composed of 100 roughly equally-sized factors will take about 434 294 481+99≈10**9 iterations. And a 1000-digit number composed of 1000 roughly equally-sized factors will take about 4+999≈103 iterations.

P and Q Have Vastly Different Bit Lengths

Reused Primes:

Assume we do reuse primes, where N1 = p*q1, N2 = p*q2

if we find gcd(N1,N2) it’ll be equal to p, breaking the entire cipher

Problems with Exponents e and d:

Small e

when e is too small, breaking RSA is as simple as finding the e-th root to the enciphered message, especially when the enciphered message isn’t subject to the modulus.

Large e

when e is too large, the encryption is usually vulnerable to 2 different attacks

1. Wieners attack
2. you find the other one ;)

e Isn’t Coprime with phi(n)

This results in all sorts of mess, in some cases the message is gone for good and can’t be decrypted, in other we can break the cipher using modular square root.

Problems with the messages M

Short Message

If our message is too short, it takes us to a similar problem as having a small e, take this as a rule of thumb to identify this issue

```
m # message
e # exponent
n # modulus
assert m**e > n
```

if this assertion fails then we have a vulnerable cipher

Bad Padding

When it comes to bad padding, attacks like Coppersmiths’ and Franklin Reiter attacks are the most common, we’ll explain further as we solve challenges

Message Reuse

Message reuse leaves us with good old CRT attacks, can you prove why?