Cryptography Mathematics and Basic Implementation

Christian Chinchole

July 16, 2023

1 RSA

The RSA algorithm is asymmetric, meaning it works with two keys: the public for encryption and the private for decryption.
Important variables are as follows:

- P,Q: are the primes.
- N: $p.q$
- e: Encryption Exponent
- d: Decryption Exponent
- N,e pair: form the public key
- N,d pair: form the private key

1.1 How are the keys generated?

1. First find the LCM of $(p-1)(q-1)$
2. $d = e^{-1} \mod \text{LCM}$
3. $n = p.q$
4. $dP = d \mod (p-1)$
   $dQ = d \mod (q-1)$
   $qInv = q^{-1} \mod p$

1.2 Pairwise Testing

The pairwise consistency test is used to check that the public and private exponent are suitable for encryption/decryption.

For k between $1 < k < (n-1$:

$$k = (k^e)^d \mod n$$

1.3 Encryption

$$c = m^e \mod n$$

1.4 Decryption

The standard method: $m = c^d \mod n$
An exponentially faster method is to use the Chinese Remainder Theorem components:

$$m_1 = c^{dP} \mod p$$
$$m_2 = c^{dQ} \mod q$$
$$h = (qInv)(m_1 - m_2) \mod p$$
$$m = m_2 + h.q$$

2 Prime Generation

Referred to SP 186-56F.3
A prime number will be generated then checked that there exists a number to satisfy

$$p^{-1} \mod e$$

It is recommended that a length of n must be atleast 1024 bits meaning that p and q will each be 512 bits respectively. According to SP800-57, the strength will be associated to the length of bits. For example 1024 $\rightarrow$ 80, 2048 $\rightarrow$ 112, and 3072 $\rightarrow$ 128. Then for M-R testing, error trials will be calculated to satisfy $2^{(-strength)}$ so for 1024 it would be 2^{-80}. It is also accepted to use 2^{-100} for all prime lengths as it was often used in the past and considered acceptable in most applications.

2.1 Integer Trial Division

Referred to SP 186-56C.7

Trial Division is recommended to only be used for ¡ 10 digit numbers.

1. Start with a Prime p to be checked.
2. Make a table of primes less than $\sqrt{p}$ (Typically done with sieving)
3. Divide p by every prime within the table, if it is divisible then fail and declare composite.
4. If it succeeds then call prime.

2.2 Sieving

Refereed to SP186-56C.8

Variables:

1. $Y_0, Y_0 + 1, ..., Y_0 + J$ where J is the final addend
2. Find a factor base of all primes p_j from 2 to a limit L. L is arbitary, but a good value is 10^3 to 10^5

Steps:

1. $S_j = Y_0 \mod p_j$ for all p_j in the factor base.
2. Initialize an array of length $J + 1$ to zero
3. Starting at $Y_0 - S_j + p_j$ let every p_j^{th} element of the array be set to 1. Do this for the entire length of the array and for every j.
4. When finished, every location in the array that has the value 1 is divisible by some small prime, and is therefore a composite.

Sieving with this method has an efficiency of approximately $M \log \log L$ where M is the length of the sieve interval. If $L = 10^5$ then the sieve will remove about 96% of all composites.

2.3 Probable Prime

1. Starting with a randomly generated number, n.
2. Perform trial division checking that the gcd of $n-1$ and the primes is equivalent to 1 excluding 2.

This is done by:

1. Loop through from 1 to number of trial divisions.
2. Create an array of moduli from mod=$n \mod primes[i]$
3. Loop through from 1 to number of trial divisions.
4. Check that the number's bit length is less than or equal to 31, the δ is not greater than unsigned long length and that the square of the prime is not larger than $n + \delta$
5. If mod[i]+δ%$primes[i] == 0$ then add 2 to δ. If $\delta >$ maxdelta then restart from n random generation. If not then redo the second loop.
6. Once this succeeds add the δ to n and confirm that the bit length of n is still the correct amount.

2.4 Auxiliary Prime Method

Having wanted to implement an ACVP self test, I need to add an auxiliary generation in accordance to SP186-4 B 3.3.6, SP186-4 C.3 (Miller Rabin none enhanced used in my implementation), and SP186-4 C.9.

Important variables to note (repeated for q):

- p, The final output prime.
- $xP1$, Random number used to find the first auxiliary prime.
- $xP2$, Random number used to find the second auxiliary prime.
- xP, Random number used for auxiliary prime generation for p.
- $pRand$, also known as X, The random for testing probable primes.

Steps for the auxiliary prime's generation (B3.3.6):

1. Generate an odd integer $xP1$, starting at $xP1$ until the first probable prime is found. Repeat this for $xP2$. $p1$ and $p2$ must be the first integers to pass primality tests in accordance to C.3.
2. Next proceed to C.9 for using the auxiliary primes in generating the final prime, p.

Steps for generating the final prime using auxiliary primes (C.9):

1. Let the inputted auxiliary primes be called, r_1 and r_2 respectively.
2. Confirm the GCD($2r_1$, r_2) = 1
3. $R = ((r_2^{-1} \mod 2r_1 * r_2) - (((2r_1)^{-1} \mod r_2) * 2r_1)$.
4. Generate a random number X such that $(\sqrt{2})(2^{nlen/2-1}) \leq X \leq (2^{nlen/2} - 1)$
5. $Y = X + ((R - X) \mod 2r_1r_2)$
6. i = 0

7. If ($Y \geq 2^{nlen/2}$, goto step 4.
8. (a) Check the primality of Y in accordance to C.3. If not prime then goto 9.
   (b) $private\_prime\_factor = Y$
9. i = i+1
10. if $i \geq 5(nLen/2))$ then failure.
11. $Y = Y + (2r_1r_2)$
12. Goto step 6.

3 RNG

3.1 PRNG

PRNGs are pseudo random number generators in that they have a deterministic algorithm typically implemented within software that will use a seed value to generate a sequence of numbers. This deterministic algorithm provides a fast way to generate 'random' numbers, but this leads to a major flaw that the sequence will be repeated provided the same seed value is provided. Another weakness is there is an element of periodicity to these numbers that is ignored now since the weakness now have such large periods. For example, Mersenne Twister MT19937 PRNG has a periodicity of $2^{19937} - 1$.

3.2 TRNG

TRNG is hardware random number generation using entropy from physical sources of some source which is then used to generate random numbers. An issue with TRNGs is their timing, as they will enumerate hardware devices or I/O their speed is not simply scalable.

3.3 CSPRNG

Due to TRNG's flaw of having low scalability and high cost of performance, a common method is to use a TRNG to seed a secure PRNG. This leads to a CSPRNG (Cascade Construction RNG). This is a software implementation; however, which leads to a vulnerability to software sided attacks. It also requires a very reliable entropy source which can be hard to obtain; even with a reliable source, if it is not able to be sampled frequently, the seeding will be less frequent losing the advantage of TRNG.

3.4 DRNG

DRNG is another hardware random number generator implementation that is in modern intel cpus. It uses processor resident entropy to repeatedly seed a hardware implemented CSPRNG. This produces high quality entropy that is able to sampled quickly, also isolation from software side attacks, meaning high quality and performance. Assembly instructions can be used to probe this.

3.5 My Implementation

Using OpenSSL's DRBG, it can be seeded with linux's syscall to get_random which makes a call to /dev/urandom which is filled with data from csprng.