Cryptography Mathematics and Basic Implementation

Christian Chinchole

August 5, 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 → 80, 2048 → 112, and 3072 → 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]+$\delta\%primes[i]$ == 0 then add 2 to δ. If δ > 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.

4 Hashes

4.1 Purpose

Hashes are used to create a condensed representation of a message that is secure in that from a hash, the message cannot be derived; i.e. one way. The numberings, SHA-*512* or SHA-*256*, stand for the maximum security and message digest size in bits.

5 SHA-1 and SHA-2

All SHA1 and SHA2 implementations were done following FIPS 180-4 document.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{64}$	1024	64	384
SHA-512	$< 2^{64}$	1024	64	512
SHA-512/224	$< 2^{64}$	1024	64	224
SHA-512/256	$< 2^{64}$	1024	64	256

5.1 Process

5.2 TODO

Add the constants and function tables for sha1, sha2.

1. SHA-1, SHA-224, and SHA-256 Process follows that a 32bit array of hash values named H will be initially loaded with predetermined H_0 values while SHA-384, SHA-512 use 64bit values for H.

2. Data will be inputted into the $\boldsymbol{updatefunction}$ where it will be padded to make sure it will be a multiple of the message size in bits. In SHA-1, 224, and 256 a 1 will be set following the message followed by k bits till the last 64 bits to allow for the length of the message. If a message is > message size bits, the first message to be processed will contain all data and the following will contain the remaining information and length. For SHA-384 and 512 there will be 128 bits for the length of the message to be used.

5.3 SHA-1 Parsing

SHA-1 uses a message scheduler of 80 32bit words and 5 working variables to process the data. W_t is the name for the message scheduler and is loading from 0-15 with M_t, message block, and from 16-79 with $ROTL^1(W_{(t-3)} \bigoplus W_{(t-8)} \bigoplus W_{(t-14)} \bigoplus W_{(t-16)}$.
Then from 0-79 in, the word blocks are processed using the function dependent upon round, round constant, and rotating the variables. After the data has been processed and put into working variables, it will then be added back to the starting hashes of the context. Refer to FIPS 180-4 6.1.2 for hash computation.

5.4 SHA-2 Parsing

SHA-2 follows a similar algorithm to SHA-1, but uses the sigma and summation functions to setup the message schedule. SHA-224, SHA-256 use 32bit words while SHA-384 and 512 use 64bit words. Refer to FIPS 180-4 6.4.2 for hash computation.

5.5 Digest

Data is digested by shifting the context's hash variables to a pointer and will always be the exact size in bits of the hash name i.e. SHA-512 will always output 512 bits of data.

5.6 Mistakes made during development

When running tests, I noticed that on every 56 byte multiple of data, there would be an extra message sent for processing and digesting. This issue occured due to me incrementing the block pointer then checking if it was greater than or equal to the maximum allowed message length. The issue was I needed it to only check for greater than cases, since using equal to would result in a message full of 0's to be processed.

6 HMAC

HMAC is a message authentication code that utilizes a hash algorithm with a key to create authentication.

6.1 Process

First we check if the key will take more than a single block to digest, if it will then we will send it's own sha update with the key length then copy it into memory (namely $outerkey$ and $innerkey$) else we will copy it to memory without performing an update. Afterwords all the $outerkey$ data will be xor'd against 0x5c and $innerkey$ 0x36. Then we will perform a digest with the inner key and message together and a digest with the outer key and previous output together. The latter's ouptut will be our final digest being padded with our key.

7 SHA-3

The SHA-3 algorithm is designed to be faster and have higher or equal security to SHA-2. It's core algorithm was designed by the Keccak team. For my implementation of SHA-3, I followed the Keccak team's pseudo code for the functions to be performed during the rounds. The implementation follows FIPS 202 guidance.

7.1 Variables

A is the state array
b is the permutation width in bits
c is the capacity of the sponge function.
d is the XOF digest length
J is the input string to rawSHAKE
l is the $log_2 w$, represents the log of the lane size
w is the lane size

7.2 Sponging the data

Unlike in previous SHA methods, SHA-3 uses sponges instead of strings as the input to the hash computation algorithm. The sponge represents the state of the data and the shape is decided by variables w and l.

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

$S = S[0]||S[1]||\ldots||S[b-2]||S[b-1]$

The state array represents the data string in a 3 dimensional space.

INSERT FIPS IMAGE HERE

$A[x, y, z] = S[(5y + x) + z]$

7.3 Functions

All functions here are represented with respect to $0 \le x \le 5, 0 \le y \le 5, 0 \le z \le 5$

7.3.1 θ function

The effect of θ is to XOR each bit in the state with the parities of two columns in the array. In particular, for the bit $A[x_0, y_0, z_0]$, the x-coordinate of one of the columns is $(x_0 - 1) \mod 5$, with the same z-coordinate, z_0, while the x-coordinate of the other column is $(x_0 + 1) \mod 5$ with z- coordinate $(z_0 - 1) \mod w$

7.3.2 ρ function

The effect of ρ is to rotate the bits of each lane by a length, called the offset, which depends on the fixed x and y coordinates of the lane. Equivalently, for each bit in the lane, the z coordinate is modified by adding the offset, modulo the lane size.

INSERT RHO OFFSET TABLE PICTURE

7.3.3 π function

The effect of π is to rearrange the positions of the lanes, as illustrated for any slice in Figure 5 below. The convention for the labeling of the coordinates is depicted in Figure 2 above; for example, the bit with coordinates x = y = 0 is depicted at the center of the slice.

INSERT FIGURE 5 PICTURE

7.4 χ function

The effect of χ is to XOR each bit with a non-linear function of two other bits in its row, as illustrated in Figure 6 below.

INSERT FIGURE 6 PICTURE

7.4.1 ι function

The effect of ι is to modify some of the bits of Lane (0, 0) in a manner that depends on the round index. The other 24 lanes are not affected by ι.

INSERT THE CONSTANT TABLE HERE

7.5 XOF Functions

SHAKE128(M, d) = KECCAK[256] (M || 1111, d)
SHAKE256(M, d) = KECCAK[512] (M || 1111, d)

8 Elliptic Curve Digital Signatures

8.1 Intro

EC-DSA utilizes either prime or binary fields to generate a key pair that will be used to sign messages. Within SP 800-186, recommended domain parameters are stated for use with both types of fields. The advantage of using EC over RSA is that EC is significantly more time consuming for a computer to break due to being a logarithmic process rather than prime factorization.

8.2 Elliptic Curve Math

All sections assume the following:
Considering a curve $y^2 = x^3 + ax + b$ on $\mathbb{Z}$
Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$

8.2.1 Identity

The point at infinity, $\mathcal{O}$, is the identity point within EC.

8.2.2 Point Negation

$(x_1, y_1) + (-(x_1, y_1)) = \mathcal{O}$
$(x_1, y_1) + (-x_1, -y_1)) = \mathcal{O}$
$(x_1, -y_1) = -(x_1, y_1)$

8.2.3 Point Doubling

This refers to using the tangent to the curve, E, at P to find the coincident point.
$\lambda = \frac{3x_1^2+a}{2y_1}$

8.2.4 Adding two points

R, the sum of P and Q, is defined as the negation of the point resulting form the intersection of the curve, E, and the straight line defined by the points P and Q.
$\lambda = \frac{y_2-y_1}{x_2-x_1}$
$x_r = \lambda^2 - x_1 - x_2$
$y_r = \lambda(x_1 - x_r) - y_1$
This requires non mutual inverses and for both points not equal to infinity.

8.2.5 Multiplying points

This is computationally done by iterating the bits in the scalar. If the bit is set, then add the result and a temporary variables, set to P at start. Then double the temp variable.

8.3 Key Pair Generation

Do tomorrow.

8.4 Signature Generation

Do tomorrow.

8.5 Signing a message

Do tomorrow.

8.6 Verifying a message

Do tomorrow.