**Prime number functionality for RSA**

**Implement the typical functions around primes and for supporting the RSA method, like a prime number test, the generation of primes, the factorization of compound numbers, exponentiation, calculations in fields, ... This is related to [C033] and should use the already implemented factorization methods. It's also related to [C034].**

RSA depends on two large random prime numbers being generated (at least 512 digits, but ideally more like 1024 digits), these digits must also be quite far apart. Math is done on the prime numbers to create a range to choose a third prime number from, this prime number is used as the public key, and it’s inverse within the range is used as the private key (<http://doctrina.org/How-RSA-Works-With-Examples.html>)

Since the security of the RSA depends on the public and private keys, and these keys are dependent on the prime number generation algorithms, these algorithms need to be able to produce large and random prime numbers.

Generating prime numbers: This is usually done by generating a large random number, and testing for prime-ness, if the random number fails the prime test, then add 1 to the original number and perform the prime test, continue this process until a number is verified to be a prime number.

Testing prime numbers: This is more difficult. An existing algorithm is the most common tactic used, the [Rabin-Miller primality tester](http://en.wikipedia.org/wiki/Rabin-Miller) (<https://en.wikipedia.org/wiki/Miller-Rabin_primality_test>). However, this is a probability test and does not give a 100% accurate response – to make the probability small enough to be considered safe to use as keys the test is performed iteratively 64 times, which results in a probability of error of only 1/(2^128).