PRIMALITY TESTING TECHNIQUES AND THE IMPORTANCE OF PRIME NUMBERS IN SECURITY PROTOCOLS

Research Assistant Enis Karaarslan, enis.karaarslan@ege.edu.tr
Ege University, International Computer Institute, 35100, Bornova-İzmir, Turkey

Abstract- This study presents primality testing by mostly emphasizing on probabilistic primality tests. It is proposed an algorithm for the generation and testing of primes, and explained Carmichael numbers. The importance of prime numbers in encryption is stated and experimental results are given.
Keywords- Prime Numbers, Carmichael Numbers, Primality Tests, Prime Generation, Security Protocols, Factorization

1. INTRODUCTION

Random and prime number generators are vastly used in many branches of science, especially in network security protocols, simulation and cryptology (public encryption). If the generated numbers are insufficient or faulty, this could lead to the failure of the application.

Large random numbers are put into primality tests in the GAP[1] environment; *division by small primes* and *addition instead of division* methods are implemented for finding primes more quickly. Also the base number needed to eliminate the first 246.683 Carmichael numbers is given for the implemented primality tests.

Security protocols use modulus n values which are the product of big (512, 1024 bit) prime numbers (p, q) as keys. The strength of these protocols depend on the hardness of factorization. It is shown that the n value, which should be the product of two prime numbers is factored easily when one of them is not a prime.

2. PRIME NUMBERS

Natural Numbers (greater than one) which can only be divided by 1 and itself are called *prime numbers*. Non prime numbers called *composite numbers*.

Prime numbers and their properties were first studied extensively by the Greek mathematicians; Euclid, Euler and Eratosthenes. Then Arabic/Islamic mathematicians like Thabit ibn Qarra, Ibn tahir Al-Baghdadi, Ibn Al-Haytham studied prime numbers [2] [3]. At the 17th Century, the studies of Fermat, Mersenne and Euler's theorems started a new century in primality research. Detailed information about the history of prime numbers can be obtained from O'Connor and Robertson's article [4].

[1] GAP (Groups, Algorithms and Programming) Software package. For details reference [1] can be used.

Security threats like Pollard P-1 and Williams P+1 factoring algorithms lead to the notion of *strong prime* which is characterized by several restrictions. Actually keys generated according to such requirements are in practice no more secure than keys that are not [5].

Prime Number Theorem states that, the number of primes less than *n* is given by Formula 1 [6]. This formula gives an idea about distribution of prime numbers.

$$\prod\nolimits_{n}^{\infty} n = \frac{n}{\ln n} \tag{1}$$

3. PRIMALITY TESTING

As prime numbers do not follow a known pattern, knowing that a number is prime or not is not an easy task. For number n to be prime, there must be no divisor of *n* between 2 and $\sqrt{n}$. For small n numbers it can be calculated, but as *n* increases it becomes not feasible. Assuming that a division takes 1 milisecond CPU time; it can be calculated that testing the first 101 digit number (1.10^{100}) is prime or not, using this division method can take about $1.9 * 10^{41}$ years CPU time. For testing large numbers for primality, more sophisticated approaches must be used. Tests can be classified into two groups:

- Deterministic Primality Tests
- Probabilistic Primality Tests

3.1. Deterministic Primality Tests

These tests can determine whether a number is prime. These tests are mostly based on factorization techniques. Two best current methods are Cyclotomic Ring Test and Elliptic Curve Test [5]. *Lucas Test* is designed to find Mersenne[2] primes and it is a deterministic primality test for Lucas sequences [7]. Deterministic tests are so complicated to implement that probability of making an error in the implementation far exceeds the probability that a probabilistic test will return composite [5].

3.2. Probabilistic Primality Tests

These tests can determine whether a number is prime or not with a given degree of confidence. Assuming this degree of confidence is large enough, these tests are good enough [8]. As probabilistic tests are fast, it is suggested to be used with error ratios smaller than 2^{-100} ($\sim 7.8 \times 10^{-31}$) [9]. Fermat, Slovay&Strassen, Lehmann, Miller&Rabin (M&R) and Frobenius tests can be given as examples of probabilistic primality tests. More theory about these tests can be found in the following references: [8], [10], [11], [12], [13], [14], [15]. Numbers passing these tests are called *probable primes* or *pseudoprimes*.

[2] Mersenne primes: For $n > 1$, prime numbers in the form of $2^n - 1$

Witness: A number between 1 and n that can be used to demonstrate the compositeness (non-primeness) of n is called *witness*. The density of witnesses (1-d) is very small. The probability of number n to be prime after i iterations is given in Formula 2:

$$\text{Prb(prime)} = 1 - \text{Prb(composite)} = 1 - (1-d)^i \qquad (2)$$

A larger d means faster convergence to the desired confidence threshold. M&R is the most popular in practice since density is large (d=0,75) [16]. To have an error ratio about 2^{-100}, 50 different base (a) values must be taken with M&R test. Actually these numbers are very pessimistic. Something like %99.9 of the possible base (a) values are witnesses [8].

Fermat's Theorem & Pseudoprime: This theorem is the basis for probable primality tests. n is called *Fermat pseudoprime* to the base a, if n satisfies Formula 3 for base a (a to be any integer $1 \leq a \leq n$-1). In other words, a *pseudoprime* is a number that "pretends" to be prime by passing Fermat's theorem for given base values [6].

$$a^{n-1} \neq 1 \pmod{n} \qquad (3)$$

Miller&Rabin(M&R) Test: The mostly used probabilistic test in practice is M&R test. It is also known as the strong pseudoprime test. The test is based on the following fact: Let the number n to be tested is an odd number. Find the s and r values with Formula 4. Let the base a be any integer such that gcd(a,n)=1**.** If the equations in Formula 5 or Formula 6 are valid, n is said to be *strong pseudoprime* according to the base a [10].

$$n - 1 = 2^s r \qquad (4)$$

$$a^r = 1 \pmod{n} \qquad (5)$$

$$a^{2^j r} = -1 \pmod{n} \quad (0 \leq j \leq s-1) \qquad (6)$$

3.3. Algorithm for Primality Testing

Pseudoprimes are very rare, at least among numbers with no small divisors [17]. Firstly division by small primes method is implemented. As there is no standard rule about the number of small primes to be used, the first 1028 primes are taken (up to 8191) as RSA EURO encryption code does.

As stated in Silverman's article [5], there is no known composite integer which passes single Miller&Rabin (M&R) test, followed by a single Lucas probable prime test. While a formal estimate of the probability of error for a combined M&R/Lucas test is still lacking, heuristics suggest that counter-examples are extremely rare [5]. It is also stated that the best test would appear to be some combination of M&R and Lucas tests [18]. Therefore Lucas test is implemented after M&R tests. Algorithm is as follows:

1. Read random data from file.
2. Make the number odd. Take the first odd number.
3. Perform division with small prime numbers (first 1028 primes) test. If it can't pass the test, this number is not a prime.
4. If n passes Step 3, do the M&R Test. If it can't pass the test, this number is not a prime.

5. If n passes Step 4, do the Lucas Test. If it can't pass the test, this number is not a prime. Else, output the number n as "probable prime".

3.4. Algorithm for Prime Generation

Firstly a Perl script is used to collect random system data from a Unix workstation for generating random sequence. This data is then put into MD5[3] function to form 128 bit random data. Consecutive runs produce larger random data that can be used as a starting point to generate a prime[19].

When generating primes, subsequent odd numbers ($n = n + 2$) are taken. *Addition instead of division* method is used for speed up. In initialization stage, residue modulo is computed by dividing with all small prime numbers. Then each time n is increased by 2, 2 is added to the current value modulo for each prime and each residue is tested for 0. Being 0 means that number is divided by the prime and subsequent odd number should be taken [6]. Algorithm is as follows:

1. Read random data from file.
2. Make the number odd. Take the first odd number.
3. Perform division with small prime numbers (first 1028 primes) test. If it can't pass the test, take the next odd number ($n = n + 2$) and go to Step 3.
4. If n passes Step 3, do the Miller&Rabin Test. If it can't pass the test, take the next odd number ($n = n + 2$) and go to step 3.
5. If n passes Step 4, do the Lucas Test. If it can't pass the test, take the next odd number ($n = n + 2$) and go to step 3. Else, output the number n as "probable prime".

3.5. Carmichael Numbers

Carmichael numbers are numbers which mislead probabilistic probability tests. Andrew Granville and Pomerance proved that for any given finite set of bases there are infinitely many Carmichael numbers that are 'strong pseudoprimes' to all the bases in that set [20]. Pinch shows that there are 246.683 prime numbers up to 10^{16}, all with at most 10 prime factors [21].

4. IMPLEMENTATION

4.1. Carmichael Numbers

The Carmichael numbers up to 10^{16} (246.683 unit)[4] are tested with different bases. The run times are calculated. Tests are implemented on Solaris Unix operating system running on Sun Ultra 5 workstation. Tests are implemented on GAP environment with Runtime() function. Runtime() function gives the used cpu time and the unit used is milisecond cpu time (msec). The implemented tests are:

---

[3] MD5 is an algorithm that takes variable length data and produces 128-bit message digest.

[4] Carmichael numbers up to 10^{16} is available by anonymous ftp ftom ftp.dpmms.cam.ac.uk in directory /pub/Carmichael. Detailed information and statistics can be obtained from Pinch's paper [21].

- *Asal_mi Test* : Division with small prime numbers (the first 1028 primes) + Lucas + Miller&Rabin Test (1 Base)
- *Miller&Rabin Test* (Given base)
- *Lehmann Test* (Given base)
- *Lucas Test*
- *IsPrime Test* (Gap native primality test)

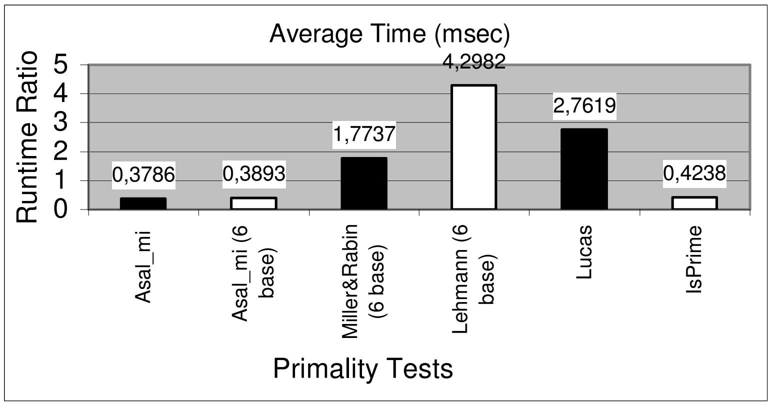
Numbers detected as primes in the test are given in Table 4.1, runtimes according to the base unit are given in Table 4.2. Average running times are given in Graph 4.1.

Table 4.1. Numbers detected as primes in the Carmichael Test

Base Unit :	**1 Base**	**2 Base**	**3 Base**	**4 Base**	**5 Base**	**6 Base**
Asal_mi	0	-	-	-	-	-
Lucas	0	-	-	-	-	-
Miller&Rabin	1818	143	27	4	2	0
Lehmann	272	88	29	10	1	0
IsPrime	0	-	-	-	-	-

Table 4.2. Runtimes According to the Base Unit (msec)

Base Unit:	**1**	**2**	**3**	**4**	**5**	**6**
Asal_mi	0,37858	-	-	-	-	-
Miller&Rabin	1,51364	1,46653	1,47127	1,74920	1,75439	1,77372
Lehmann	1,48867	2,31479	2,93830	2,93830	3,91316	4,29823
Lucas	2,76187	-	-	-	-	-
IsPrime	0,42382	-	-	-	-	-



Graph 4.1: Average Running Times in the Carmichael Test

4.2. Factorization of Numbers

Factorization is implemented with FactInt 1.1[5] package in GAP environment. It is obvious that better factorization methods are available but the purpose of this study is to show the hardness of factorization of number *n,* which is the product of two prime numbers (*p*, *q*). Factorization time of *n,* which is the product of two primes is given in Table 4.3; factorization time of *n,* which is the product of one prime and one nonprime is given in Table 4.4. The experiments are implemented with 100 samples (*p*, *q* pairs). If the average factorization time of a number which is the product of one prime and one nonprime is taken as 1 unit, the average factorization of a number which is the product of two primes is given in Table 4.5.

Table 4.3. Factorization Time of *n* which is the Product of Two Primes (msec)

Bit Length of Factors (*p*, *q*)	**16 bit**	**32 bit**	**64-bit**	**96 bit**
Min Time	0	80	5.350	2.555.480
Max Time	40	9.750	123.380	11.475.550
Avarage	10,12	1.614,93	67.619,31	6.745.135

Table 4.4. Factorization Time of *n* which is the Product of One Prime and One Nonprime (msec)

Bit Length of Factors	**16 bit**	**32 bit**	**64 bit**	**96 bit**
Min Time	0	10	60	4.920
Max Time	30	8870	85.770	4.156.480
Avarage	3,23	242,33	19.479,54	934.243

Table 4.5. Comparision of Factorization Times

Bit Length of Factors	**16 bit**	**32 bit**	**64 bit**	**96 bit**
One Prime	1	1	1	1
Two Prime	3,133127	6,664176949	3,471299117	7,219893539

[5] FactInt 1.1 (Routines for Integer Factorization), is the factorization package written by Stefan Kohl for the GAP environment. More info and source code can be obtained at http://www-gap.dcs.st-andrews.ac.uk/~gap/Share/factint.html

5. DISCUSSION & CONCLUSION

- Security protocols use modulus n values, which are the product of big (512, 1024 bit) prime numbers (p, q) as keys. It is shown that n value, which should be the product of two prime numbers is factored easily when one of them is not a prime. This means that with that value used, it would be easier to find the key values and break into the system. Because of these, primality testing and prime generations techniques are very important.
- Deterministic Primality Tests take too much time to implement especially for large numbers. Probablistic Primality tests are preferred in practice.
- For primality testing we suggest the algorithm represented in section 3.3. For prime generation we suggest the algorithm represented in section 3.4.
- As it can be seen from the results of section 4.1, *Asal_mi* and *IsPrime* tests take less time than the others. This shows that "division by small primes" speed up the prime finding process as composite numbers including Carmichael numbers have prime factors.
- Carmichael numbers mislead probabilistic probability tests. It can be seen in section 4.1 that at least 6 bases must be taken with probabilistic primality tests for numbers up to 10^{16} to eliminate Carmichael numbers.

There are many problems to be solved in the area of prime numbers. It should not be forgotten that studies about prime numbers lead to developments in the Number Theory, mathematics and many sciences.

REFERENCES

[1] GAPRef, GAP Release 4.1 - Reference Manual, The GAP Group, School of Mathematical and Computational Sciences University of St. Andrews, 1999.

[2] MacTutor, The MacTutor History of Mathematics Archive, http://www-groups.dcs.st-and.ac.uk/history/, 2002

[3] O'Connor, J.J. and Robertson E.F, Arabic Mathematics: Forgotten brilliance?, http://www-history.mcs.st-andrews.ac.uk/history/HistTopics/Arabic_mathematics-.html, 1999.

[4] O'Connor, J.J. and Robertson E.F., History of Prime Numbers, 1996. http://www-history.mcs.st-andrews.ac.uk/history/HistTopics/Prime_numbers.html

[5] Silverman, R.D., Fast Generation of Random, Strong RSA Primes, RSA Laboratories' Crypto Bytes Magazine - Volume 3, Number 1, 1997.

[6] Penzhorn W.T., Fast Algorithms For The Generation of Large Primes For The RSA Cryptosystem, 1992.

[7] Emerson P., Prime Number Generation and Primality Testing, MSc Thesis, Computer Science at Middlebury College, 1997.

[8] Schneier B., Applied Cryptography (Second Edition), John Wiley & Sons Inc., pages: 258-261, 1996.

[9] RSA, RSA FAQ v4, Frequently Asked Questions About Today's Cryptography – What's Primality Testing?, http://www.rsasecurity.com/rsalabs/faq/2-5-1.html, 1998.

[10] Menezes, A. and Oorschot P., Handbook of Applied Cryptography, CRC Press, 1997.

[11] Caldwell C.K., Finding Primes & Proving Primality, http://www.utm.edu/research/primes/prove1.html, 1997.

[12] Zachary S, McGregor Dorsey, Methods of Primality Testing, http://www-math.mit.edu/phase2/UJM/vol1/DORSEY-F.PDF , 1999

[13] Grantham, J., Frobenius Pseudoprimes, Institute for Defense Analyses, Center for Computing Sciences, 1998.

[14] Grantham, J., A Probable Prime Test with High Confidence**,** Journal of Number Theory 72, 32-47, 1998.

[15] Maurer, U.M., Fast Generation of Prime Numbers& Secure Public-Key Cryptographic Parameters, to appear in Journal of Cryptography, 1994.

[16] Segre, A., Computer and Network Security, Iowa University "Data Security" Lecture Notes (last modified 97), 2000.

[17] Rivest, R., Finding Four Million Large Random Primes, Advances in Cryptology, CRYPTO'90, LNCS 537, pages: 625-626, 1990.

[18] Pinch, R.G.E., Some Primality Testing Algortihms, Proc 4th Rhine workshop on Computer Algebra, Karlsruhe, www.chalcedon.demon.co.uk/rcam.html, 1994

[19] Karaarslan, E., Large Random & Prime Number Generation, Msc. Thesis, Ege University International Computer Institute, 2001**.**

[20] Granville A., Primality Testing & Carmichael Numbers, Notices Amer. Math. Soc. 39 (pages: 696-700), 1992.

[21] Pinch, R.G.E., The Carmichael Numbers Up to 10^{16} www.chalcedon.demon.co.uk/rcam.html, 1998.