Research Journal of Applied Sciences 10 (8): 358-364, 2015
ISSN: 1815-932X

Investigation of Distribution of Pseudosimple Numbers

Bulat G. Mubarakov, Andrey S. Mochalov and Ramilya G. Rubtsova
Kazan Federal University, Kremvleskays 18, Kazan, Russia

Abstract: Prime numbers play an important role in modern cryptography. The system of RSA is one of the most well-known cryptographic systems. It uses compound number as a key that is the product of two prime numbers of large dimension. To search for large prime numbers different algorithms of primality tests are used. Note that all existing algorithms of primality testing are divided into two large groups: deterministic and probabilistic tests. Deterministic simplicity tests can accurately say whether, the number is prime or composite. The algorithms of the second group are much faster than deterministic but operate with a certain probability of error. Among the tests of simplicity used in cryptography, the most popular is the probabilistic polynomial primality test of Miller-Rabin. The theoretical probability of an error of this test depends on the number of iterations wherein each iteration decreases the probability of error in 4 times. However, the actual number of errors made by this test is much less than the theoretical. For example, checking simplicity of all odd numbers less than one million with one iteration of Miller-Rabin test with the base of two gives 47 errors with total number of prime numbers of this range of about 78 thousand. We note that the test of Miller-Rabin makes mistakes of one type: it defines some composite numbers as simple. These numbers are called strictly pseudo prime according to the specified reason. The study of the distribution of strictly pseudo prime numbers for various reasons allows to accelerate execution of Miller-Rabin test. This study describes one way to improve the efficiency of the probabilistic Miller-Rabin test. The theoretical basis for this is to search and study the properties of the distribution of strictly pseudo prime numbers.

Key words: Cryptography, primality test, the test of simplicity of Miller-Rabin, pseudo prime numbers, rang

INTRODUCTION

One of the most important problems in cryptography and number theory is the problem of determining by the set of natural numbers n whether, it is prime or compound. Thus, a well-known cryptographic system with public key RSA composite number n is used as a product of two prime numbers p and q. The successful solution of the problem of the factorization (decomposition) of n allows to hack RSA scheme and to determine a secret key, then, the choice of p and q, largely determines the strength of the encryption (Rivest, 1978). For the definition of simplicity, there are many different tests of simplicity of which the most common is the Miller-Rabin test of simplicity (Rabin, 1980; Crandall and Pomerance, 2005; Ishmukhametov, 2014; Ishmukhametov and Mubarakov, 2013). Miller-Rabin test works quite quickly and efficiently. However, the main problem with this test a determination of the number of iterations required to guarantee determination of checked number simplicity. Since, there are composite numbers that successfully pass the test of Miller-Rabin, they are called strictly pseudo prime (Pomerance *et al*., 1980a, b; Jaeschke, 1993; Jiang and Deng, 2012). Thus, analysis of the search algorithms of strictly pseudo prime numbers and study of the properties of distribution of strictly pseudo prime numbers can increase the speed of the Miller-Rabin test and improve its accuracy.

In this research, we consider another way to increase the efficiency of the test of Miller-Rabin and analysis of search results of strictly pseudo prime numbers.

THE ALGORITHM OF RSA

In 1978, three American Scientists, Ronald Rivest, Adi Shamir and Leonard Adleman of MIT proposed a new idea of encryption using two different keys: the open (public) and private (closed) for encoding and decoding text messages. This method is called RSA Method, the first letters of the names of the researchers of this method. In connection with tremendous growth of local and global computer networks RSA Method was further spread in a wide variety of products across different platforms and in many industries (Ishmukhametov and Rubtsova, 2007). Currently, RSA cryptosystem is built into many commercial products, whose number is constantly increasing. In addition, it is used by the operating system Microsoft, Apple, Sun and Novell. In hardware, RSA

Corresponding Author: Bulat G. Mubarakov, Kazan Federal University, Kremvleskays 18, Kazan, Russia

algorithm is used in secure phones, Ethernet network cards on smart cards, it is widely used in the cryptographic equipment. Furthermore, the algorithm is included in all basic protocols for secure Internet communications, including S/MIME, SSL and S/WAN. With the help of this method encrypted passwords for database access, credit card numbers to pay for services and many more are sent. In addition, this method is based on the idea of the digital signature which is supported by the appropriate legislative decrees in many countries, including Russia. About 500 million users around the world use RSA BSAFE encryption technology. Since in most cases RSA algorithm is used, it can be considered the most common cryptosystem in the world and this amount has a marked tendency to increase with growing internet.

In this algorithm, prime numbers play a fundamental role. Since, you need to generate a pair of public and secret keys, it is necessary to find two large prime numbers p and q and calculate the product n = pq. Then take the required random number e, relatively prime to $\varphi(n) = (p-1)(q-1)$ and find a number d of condition $ed \equiv 1 \pmod{\varphi(n)}$. The pair (n, e) is declared a public key. The remaining numbers (p, q, $\varphi(n)$, d) form a secret key. To decrypt it is enough to know the couple (n, d). Since, the successful solution of the problem of number n factorization (decomposition) (usually 1024 or 2048 bits) allows you to completely decode the RSA scheme and determine secret key, selection of prime numbers p and q, largely determines the strength of the encryption.

Classification of number testing algorithms for primality: The procedure for determining for the given natural numbers n, whether, it is simple or compound is called a test of simplicity. To determine the ease of p and q it is known many different primality test (Ishmukhametov, 2014). All the existing algorithms for testing primality can be divided into two categories:

The so-called deterministic tests for ease which belong to the class of algorithms that determine in polynomial time whether a given number is prime or not.

Probabilistic tests that are included in the class of algorithms that determine in polynomial time whether a given number is prime with a certain probability. Thus, at the expense of algorithm execution time one can achieve high probability as much as you please.

Deterministic tests of number simplicity: There are many techniques that allow us to determine with a probability that equals one that is reliably authentic whether the given positive integer number n is simple. Unfortunately, some of them research well only for small values of n; some require a complete factorization of n-1 number and some only allow you to check numbers of a special type.

Probabilistic tests: Today, the most commonly used is probabilistic test for simplicity. It is proved, primarily by a much smaller amount spent on test as compared with deterministic tests. It is important to know that this approach is most commonly used in practice.

Requirements for high cryptographic system firmness call to practice probabilistic tests with a high degree of certainty. More recently, it was very difficult because of the imperfection of the technical characteristics of computing systems. However, performance growth of computing systems, along with an increase in the efficiency of the developed algorithms make decryption task realized.

The majority of modern probabilistic methods of primality testing is based on different variations of Fermat's small theorem. According to Fermat's small theorem if the integer n the simple, then for any integer a, not divisible by n, there is comparison $a^{n-1} \equiv 1 \pmod{n}$. From the same theorem implies that if this equation is not solvable at least for one number a in range of {1, 2, ..., n-1}, then n composite number. The inverse proposition is not true.

The task of increasing the reliability of probabilistic tests has become urgent for cryptography, after "Carmichael numbers" being found and it became clear that the test that is based on Fermat's small theorem "gives failures". The algorithm proposed by Lehmann became the first algorithm that lacks the disadvantages of Fermat's test. This test could determine the numbers of Carmichael and it cast them as components than they actually are. However, probabilistic Miller-Rabin test gives higher accuracy.

TEST OF MILLER-RABIN

Let's start with a description of the algorithm. Given a positive odd number n>3. Imagine the number of n-1 in the form of:

$$n-1 = 2^s \times u, \ u\text{-odd} \quad (1)$$

Miller-Rabin algorithm consists of several iterations called rounds. Each round either determines the number n as a compound or enhances the probability of that n is prime. We describe the implementation of the instruction of the round:

- Select a random integer a in the range from 2 to n-1
- We calculate $b = a^u \bmod n$
- We check the condition

$$b = \pm 1 \bmod n \quad (2)$$

- If Eq. 2 is executed, then the number n is probably simple go to the next round
- Otherwise, we calculate the sequence:

$$b_0, b_1, ..., b_{s-1} \text{ assuming } b_0 = b, b_{i+1} = b_i^2 \bmod n \quad (3)$$

If none of the members of the sequence is equivalent to -1 on modul n, then we claim that n composite. Otherwise, it confirms n probably simple. Miller-Rabin test works quickly and efficiently but as previously mentioned, there are components that successfully pass the test of Miller-Rabina.

Definition 1: Let a>1 arbitrary integer. Let us call odd composite integer n strictly pseudoprime on the basis of a or spsp(a) if the number n passes another round of Miller-Rabin test with the base a. In other words, one of the following conditions is met:

$$a^u \equiv \pm 1 \bmod n$$

$$a^{u2^i} \equiv -1 \bmod n \text{ for some } i, 0 \le i < s$$

Where, $n-1 = 2^s u$

Definition 2: Let a> 1 arbitrary integer. Let us call odd composite integer n a-pseudoprime or psp(a) if the following formula is executed:

$$a^{n-1} \bmod n = 1$$

The element a is called a base for pseudoprime number n. By Rabin theorem probability that a composite number successfully passes one round of this test does not exceed ¼, therefore, successful performance of k iterations reduces the probability of error to 4^{-k}. Note that if n a composite, one of the iterations will determine this accurately. If you use unproven Riemann hypothesis about the distribution of prime numbers, you can limit the number of iterations with value $(\log_2 n)^2$ that however is obviously overestimated.

Thus, an effective search of strictly pseudo prime numbers and study of their properties can increase the speed of the Miller-Rabin test and improve its accuracy.

In particular in (Pomerance *et al*., 1980a, b), it is proposed a "fast" primality test for numbers $n<25\times10^9$, the idea of which is as follows.

We will find a list of all the numbers $p<25\times10^9$ such that $p \in spsp(\{2, 3, 5\})$. Suppose that an integer n is taken to input. We define an algorithm using the following steps:

Run the test of Miller-Rabin in three iterations with witnesses of simplicity a = 2, 3 and 5 correspondingly. If n does not pass at least one of the tests, consider it an integral and complete work.

We check to see if the number is in our list ($n \in P$). If the number was found, it is evidently composite. Otherwise, we believe n simple.

It is easy to prove that the proposed algorithm is deterministic and will give correct result for any number from the interval in question. Its applicability is particularly the case in the computing systems having a sufficient amount of free memory (it is necessary to constantly keep a list P) but they may have a deficit of computing power.

PSEUDOPRIME NUMBERS SEARCH ALGORITHM

In order to find all pseudo prime numbers that are less than some fixed boundary, it would be wise to search for the numbers that are pseudo prime immediately on the basis of several numbers, especially if this will help increase the efficiency of the algorithm.

We fix some set $v = (a_1,...,a_l)$, B border and set objective in order to find all such numbers $n<B$ that $n \in spsp(v)$. To begin with we will consider only the numbers of the form n = pq, where p, q primes. The resulting approach can be extended in the future on any number of factors t but the task itself is less relevant because overwhelming majority of strictly pseudoprime numbers on several bases consist of two factors. Among the given in (Crandall and Pomerance, 2005) list of spsp{2,3,5} numbers $<10^{12}$, only six of the 101 (5.9%) have t = 3. Numbers consisting of 4 or more factors in this list are not present.

It should also be understood that at t>2, it becomes much easier to make a complete listing of multipliers as the number of options under consideration is limited to $p_i < \sqrt{B}$ for all j, $1<j<t$.

By condition that n = pq is pseudoprime on the basis b only if $p|b^{q-1}-1$ and $q|b^{p-1}-1$. If we are looking for $n \in spsp(w)$, $w = (b_1,...,b_l)$, $w \subseteq v$, we must have $q| b_i^{p-1} -1$, $1<i<l$ where:

$$q \left| НОД(b_1^{p-1} - 1, ..., b_l^{p-1} - 1) \right. \quad (5)$$

Equation 5 gives us the opportunity, fixing another factor $p<\sqrt{B}$ to find the corresponding value of q among the divisors of calculated GCD. Numbers $b_i^{p-1} -1$ will have big values but their GCD will be considerably less which facilitates factorization. Obviously, this approach would be most effective at small values of p in contrast for example, techniques proposed by Crandall and Pomerance (2005). Below we describe the algorithm:

- Choose another simple $p<\sqrt{B}$ and compute:

$$h=НОД(b_1^{p-1}-1,...,b_l^{p-1}-1)$$

- We factor $h/p = h_1h_2...h_m$
- For each $h_i>p$, we check whether h_ip is strictly pseudoprime for v and o to step 1

Obviously, the effective implementation of the algorithm depends on the following tasks:

- The l integers are given, you need to calculate their GCD
- For the h/p to find all the factors

We can assume that among the divisors h/p there will be a sufficient number of small primes in addition as will be shown below the value of h/p is small even when $p\approx10^7$. Accordingly, the use of such sub exponential factorization algorithms as a method of quadratic sieve or the number field Sieve Method will be unjustified. These algorithms require a significant amount of time at the stage of initialization (for example, the choice of a polynomial pair in the number field sieve method) and are most effective when factoring numbers $>10^{100}$.

In this case, choice of such factorization methods as ρ-algoritm of Pollard and the method of elliptic curves look more reasonable. The computational complexity of the latter is estimated as:

$$\exp\left(\left(\sqrt{2}+o(1)\right)\sqrt{p\ln p\ln(\ln p)}\right)$$

where, p the smallest divisor of h/p. In practice, we will first choose possible divisors $<10^3$. This will discard all small prime factors h/p without spending a lot of time on this process.

The calculation for each type numbers algorithm iteration $L(p) = b^{p-1}-1$ also seems expensive task, moreover with the size of w increasing the number of such computations will be directly proportional to the increase.

In this case, recurrent approach will be more efficient for calculation L(p). Suppose, we have calculated the value of $L(p_{i_1})$ where p_{i_1} a regular prime number. We need to quickly calculate $L(p_{i_2})=L(p_{i_1}+\delta)$ where $\delta=p_{i_2}-p_{i_1}$ some number. We have:

$$L(p_{i_1}+\delta)=b^{p_{i_1}+\delta-1}-1=b^{p_{i_1}-1}\times b^{\delta}-1=\left(b^{p_{i_1}-1}-1\right)\times b^{\delta}+\left(b^{\delta}-1\right)=b^{\delta}L(p_{i_1})+\left(b^{\delta}-1\right)$$

We find that:

$$L(p_{i_1}+\delta)=b^{\delta}L(p_{i_1})+\left(b^{\delta}-1\right)$$

Obviously, the use of the recursion formula computationally simplifies our task as we get rid of the need to perform exponentiation. As far as, the practical application is concerned, the use of the approach has reduced the running time by an average of 8%.

Calculating the GCD numbers of the form $b_i^{p-1}-1$ is a very demanding task as arguments can take large values. For example if b = 2, $p\approx10^7$, this value can be estimated as $\approx 2^{10^7}\approx10^{3\times10^6}$. Simple methods for finding the GCD as Euclid's algorithm in this case will be useless. A comparative analysis of the implementation of various methods for finding the GCD applied to our conditions showed that the most effective is to use Lehmer's GCD algorithm.

Implementation details: As the main development language, we will use C++ as a development environment, we will use Linux Operating System and NetBeans IDE.

Since, developed algorithms have originally been designed to work with large numbers that are not able to fit in the memory area, equal to the standard machine word (32 or 64 bit), there is a need to use the library long arithmetic. As such we will use the GMP an open-source software designed for floating point, integer and rational numbers with arbitrary precision calculations.

Note that the proposed search algorithm spsp(v) numbers there are no dependencies between computations produced for two different factors p, except for the recurrence formula linking consecutive numbers $L(p_i)$. Failure to use this correlation will allow you to search spsp(v) numbers of disjoint intervals p independently of one another.

Sequential calculations in this case are absent and this means organized similarly parallelization will provide nearly ideal acceleration time. According to Amdahl's Law, the acceleration that can be obtained on a computing system of k processors as compared with a single-processor solution will not exceed the value:

$$S_k=\frac{1}{\alpha+\frac{1-\alpha}{k}}$$

where, α the proportion of the total volume of calculations which can be obtained only by successive calculations in our case ≈0.

Let us aim to create an implementation of the algorithm adapted for use in a multiprocessor system with shared memory having k processors. Ideally such an implementation should provide acceleration $S_k = k$ compared with a conventional implementation.

Calculation flows should be organized in such a way that for every prime p all the relevant calculations were performed only once. Synchronization between the flows thus calculated must occur on the current p. To solve the problem, we can guarantee that at any given time only one thread has right to read/write access to variable that stores the next value p. In pseudocode such synchronization will look this way:

- We grab variable p
- We copy the value from p to the local variable p of the stream and we assign value of the next prime number p←m, m>p, m simple to the variable p
- We free the variable p
- We perform calculations and derive spsp(v) numbers where one of the factors is p. Proceed to 1

The disadvantage of this approach is the loss of CPU time spent on synchronization: with a large number of streams, some of them will be idle, waiting for blocking p to be lifted.

It is much more efficient in this case to allocate small intervals of length l [(y-1) 1; y1], y∈N for flows. Within its range, each thread will perform calculations without the need for synchronization. Moreover, there is a possibility of using the recurrence relation. Transform appropriate description of the algorithm:

- Grab the variable y
- Copy the value from y to the local variable y of the stream and execute y←y + 1
- Free the variable y
- For each p∈[(y-1)l; yl], we perform calculations according to Eq. 5, search and derive spsp(v) numbers where one of the factors is p. Proceed to 1

The correct choice of l interval length is determined primarily by the search boundary B and the number of the processors k. The maximum efficiency of the algorithm can be reached at l = B/k.

For parallelization we will use OpenMP an open standard that describes a set of compiler directives, library routines and environment variables intended for programming of multithreaded applications on multiprocessor systems with shared memory.

Having carried out the parallelization, we will run computation on a computer that has a dual-core processor. On the test range the algorithm worked for 261 sec while the sequential version of the algorithm on the same computer works off for 510 sec. That is in this case, the practical value $S_k \approx 1.954 \approx k = 2$ which confirms the effectiveness of the implementation.

SUMMARY

The complete cycle of the algorithm took 2 days 23 h 34 min and the result of his work was the list of P from 1118 numbers that are strictly pseudoprime with respect to the first three prime numbers (2, 3, 5). All numbers of this range consist of two factors where one of them is limited on the value with number 10^7. The smallest resulting number $25326001 \approx 2.5 \times 10^7$, the biggest $3931205905188049 \approx 3.9 \times 10^{15}$. Figure 1 illustrates the distribution of the numbers obtained.

In the range of 10^{12}, we find complete agreement with 95 pseudoprime numbers given (Crandall and Pomerance, 2005) in his research. Despite the fact that this is only 8.5% of the entire set of numbers of the range P, complete coincidence confirms the hypothesis of a small spread between the values of p and q and this excludes the possibility to "miss" strictly pseudoprime number fixing a specific limit for one of the factors. More detailed analysis of the spread between the factors will be made below.

We also note that this list does not contain strictly pseudoprime number defined by Crandall and Pomerance (2005) as:

$$\psi_7 = \psi_8 = 341550071728321 = \\ = 10670053 \times 32010157$$

Our list is formed for strictly pseudoprime numbers in relation to the first three prime numbers (2, 3, 5) and therefore, the abovementioned number must meet the criteria. Note, however that the least of the factors of the number $10670053 > 10^7$ and thus, it was outside the search.

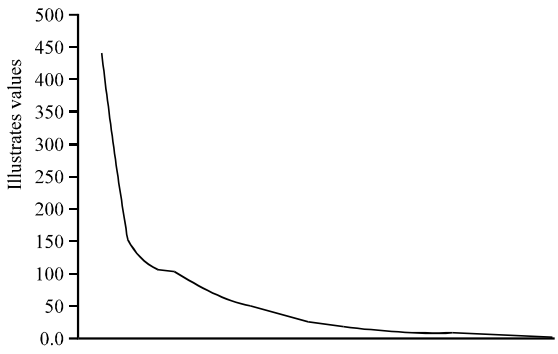

Fig. 1: The illustrates the distribution of the number

We increase the upper limit for the factor by 1% and carry out the same search again in the range of $10^7<p<1, 1\times10^7$. The calculation took another 23 h and the algorithm has issued a list of 87 numbers. Numbers $\psi_7=\psi_8$ also present in the list that confirms the correctness of the research.

Pomerance *et al.* (1980a, b) in his research, he notes that most of the investigated pseudosimple numbers can be represented in the form n = (l+1)(kl+1) where l+1 is prime, a is some small positive integer. The statement also applies to numbers consisting of three simple factors. For example, spsp{2,3,5} number:

$$3215031751 = 151\times751\times28351 = (k+1)(4k+1)$$

where, 4k+1 = (m+1)(5m+1), k = 28350, m = 150. Based on this feature, the researcher proposes to solve the problem of factorization of knowingly pseudoprime number initially with attempts to factor form (l+m)(kl+1) and only in the event of failure to move to more severe methods. The researcher also claims that due to the requirement of a strong pseudoprimeness amount of numbers that do not meet requirements of the given form will be rapidly reduced.

The researcher of Jaeschke (1993), on the contrary is trying to contest this hypothesis, citing as a counter example two pseudoprime numbers from the range $<10^{12}$ that do not meet requirements of the (l+1)(kl+1) form:

$$177475820141 = 176041\times440101$$
$$183413388211 = 370891\times494521$$

The study of numbers from the resulting list for compliance with the given form showed that 23 of 1118 strictly psevdoprime numbers do not correspond to it (2.05%).

For greater understanding of distribution of the numbers we will display them in a graph in which the vertical axis will show the number of strictly pseudoprime numbers that do not meet requirements of the above mentioned shape and smaller of this value x.

From Fig. 2, we see that with an increase of values the number of such numbers becomes much less. The last statement is an additional argument for the use of the proposed in Pomerance *et al.* (1980) approach to factoring pseudoprime numbers.

Analysis of the values of k in the formula deserves special attention. The researcher claims that these values are small. The biggest value obtained in this case, k = 61. However, ~80% of the values are concentrated in the range $2\le k\le5$. In general, the distribution of values of k for the selected numbers can be represented as a histogram (Fig. 3).

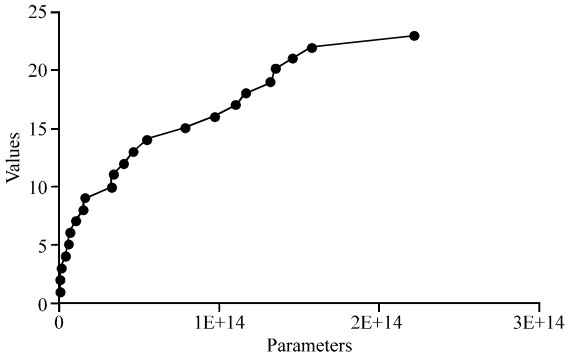

Fig. 2: The increase of values the number

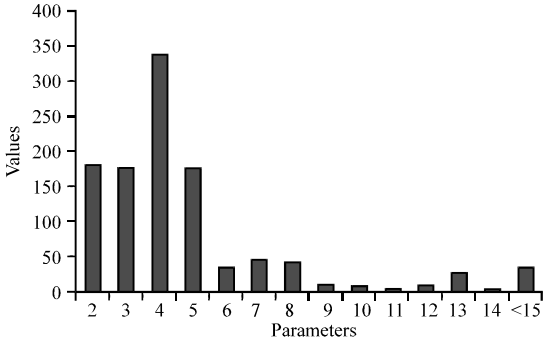

Fig. 3: The values of k for selected number be represented as histogram

CONCLUSION

The results obtained in this study may be useful both during construction of efficient simplicity tests and evaluation of errors in Miller-Rabin test and within the further study of properties of pseudoprime and strongly pseudoprime numbers.

In particular, the above mentioned list of strictly pseudoprime numbers on bases (2, 3, 5) has been used by the researchers for implementation of "quick" test of simplicity, proposed in research (Pomerance *et al.*, 1980a, b). Depending on the selected base and the size of input number one was able to gain on the average 6% acceleration as compared to the conventional implementation of Miller-Rabin test which confirms the relevance and practical value of the work.

In addition, the identified properties of pseudoprime numbers open up additional opportunities for building a more efficient algorithms of their search.

ACKNOWLEDGEMENT

The research is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

REFERENCES

Crandall, R. and C. Pomerance, 2005. The prime numbers: a computational perspective. 2nd Edn., Springer-Verlag, Berlin, pp: 604.

Ishmukhametov, S.T. and R.G. Rubtsova, 2007. On the complexity of integers factoring. Bulletin TSHPU, pp: 9-10.

Ishmukhametov, S. and B. Mubarakov, 2013. On practical aspects of the Miller-Rabin Primarily Test. Lobachevskii J. Math., 3: 13.

Ishmukhametov, S.T., 2014. Methods of integers factoring. S.T. Ishmuhametov (Eds.), LAP Lambert Academic Pub., ISBN: 978-3-659-17639-5, pp: 27-29, 256.

Jiang, J. and Y. Deng, 2012. Strong pseudoprimes to the first 9 prime bases. ArXiv: 1207. 0063 v1 [math.NT], 2012. http://arxiv.org/pdf/1207.0063.pdf.

Jaeschke, G., 1993. On Strong Pseudoprimes to Several Bases. Math. Comput., 61: 915-926.

Pomerance, C, C. Selfridge and S. Wagstaff, 1980a. The Pseudoprimes to 25×109. Math. Comput., pp: 1003-1026.

Pomerance, C., J. Selfrifge and S. Samuel, 1980b. The Pseudoprimes to 25×10^9. Math. Comput., 35: 1003-1026.

Rabin, M., 1980. Probabilistic algorithm for testing primality. J. Numb. Theory, 12 (1): 128-138.

Rivest, R.A, 1978. Method for Obtaining Digital Signatures and Public-Key Cryptosystems. R. Rivest, A. Shamir and L. Adleman (Eds.). Communications of the ACM, 21 (2): 120-126.